RL-TR-94-93
Final Technical Report
July 1994

FAULT TOLERANCE OF NEURAL NETWORKS

Syracuse University

Kishan Mehrotra, Chilukuri K. Mohan, and Sanjay Ranka

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

Rome Laboratory
Air Force Materiel Command
Griffiss Air Force Base, New York

This report has been reviewed by the Rome Laboratory Public Affairs Office (PA) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

RL-TR-94-93 has been reviewed and is approved for publication.

APPROVED: Daniel J. Burns

DANIEL J. BURNS
Project Engineer

FOR THE COMMANDER: John J. Bart

JOHN J. BART
Chief Scientist, Reliability Sciences
Electromagnetics & Reliability Directorate

If your address has changed or if you wish to be removed from the Rome Laboratory mailing list, or if the addressee is no longer employed by your organization, please notify RL (ERDR) Griffiss AFB NY 13441. This will assist us in maintaining a current mailing list.

Do not return copies of this report unless contractual obligations or notices on a specific document require that it be returned.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (Leave Blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	July 1994	Final Feb 92 - Aug 93

4. TITLE AND SUBTITLE
FAULT TOLERANCE OF NEURAL NETWORKS

5. FUNDING NUMBERS
C - F30602-92-C-0031
PE - 62702F
PR - 2338
TA - 01
WU - PF

6. AUTHOR(S)
Kishan Mehrotra, Chilukuri K. Mohan, and Sanjay Ranka

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Syracuse University
School of Computer & Information Science
Syracuse NY 13244-4100

8. PERFORMING ORGANIZATION REPORT NUMBER
N/A

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Rome Laboratory (ERDR)
525 Brooks Road
Griffiss AFB NY 13441-4505

10. SPONSORING/MONITORING AGENCY REPORT NUMBER
RL-TR-94-93

11. SUPPLEMENTARY NOTES
Rome Laboratory Project Engineer: Daniel J. Burns/ERDR/(315) 330-4055

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT (Maximum 200 words)

This effort studied fault tolerance aspects of artificial neural networks, and resulted in the development of neural learning techniques that more effectively utilize the inherent redundancy and excess of resources over the minimum required found in most classically trained networks. Performance evaluation measures were developed and used to quantify network tolerance to faults such as single link failures, multiple node failures, multiple link failures, and also small degradations in multiple links or nodes. Several variations of the basic back-propagation algorithm were designed and implemented, yielding improvements in fault tolerance. An "Addition-Deletion" algorithm was designed to successively modify the size of a network by deleting nodes that do not contribute to fault tolerance, and to add new nodes in a way that is assured to improve fault tolerance. The techniques designed in this project were compared to those suggested by others, and were found to improve robustness. Also, a "Refine" algorithm was defined, which takes a robust network which does not satisfy hardware restrictions and transforms it to another network which does.

14. SUBJECT TERMS	15. NUMBER OF PAGES
Artificial Neural Networks, Robustness, Reliability, Fault Tolerance, Training Algorithms	110
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL

WORK SUMMARY

The following tasks were proposed and carried out for USAF Contract No. F 30602-92-C-0031.

(1) **Performance evaluation and analysis of fault tolerance of existing neural networks.** Different evaluation measures may be needed for neural networks intended to perform different tasks. Various sensitivity measures were developed, modeling many different kinds of faults: single node failures, single link failures, multiple node failures, multiple link failures, and also small degradations in multiple links or nodes.

(2) **Designing new fault tolerant training algorithms for feedforward neural networks.** Several variations of the basic backpropagation algorithm were designed and implemented, yielding improvements in fault tolerance. The learning rule was modified, penalizing higher magnitude weights.

(3) **Network modification on detecting faults.** The "Addition-Deletion" algorithm was designed to successively modify the size of a network by deleting redundant nodes that do not contribute to fault tolerance, and to add new nodes in a way that is assured to improve fault tolerance.

The techniques designed in this project were compared with alternative methods suggested by other researchers, and found to improve robustness. In addition to these tasks, research was also carried out on methods of extending these methods to hardware implementations of neural networks. An algorithm "Refine" was defined, which takes a robust network that does not satisfy hardware restrictions on magnitudes of various network parameters and transforms it into another network that does satisfy hardware constraints.

Fault Tolerance of Neural Networks

Kishan Mehrotra, Chilukuri Mohan, Sanjay Ranka
School of Computer and Information Science
Syracuse University, NY 13244-4100

1 Problem Description

Neural networks[1] have been used in various applications, including classification tasks, pattern recognition, image processing, expert systems, and adaptive control systems. In recent years, various hardware implementations of neural networks have also been developed. Since neural networks often contain a large number of computational units, in excess of the minimum required, they have considerable potential for fault tolerance. However, there has been very little systematic study of the fault tolerance of neural networks, and classical neural learning algorithms like backpropagation do not make an attempt to develop fault tolerant neural networks. Using neural networks with no built-in or proven fault tolerance can lead to disastrous results when localized errors occur in critical parts of neural networks. Therefore, it is important to study the reliability aspects of neural networks, and develop neural learning techniques that effectively utilize the redundancy inherent in neural networks. Applications based on such networks will be far more reliable than less carefully designed neural networks.

1.1 Fault Tolerance

Fault tolerant computing systems [2, 15, 18, 12] are those that produce correct results or actions even when some of their components and subsystems fail; this is often achieved by introducing redundancy and error-correcting mechanisms. A system has potential for fault tolerance if the amount of resources used by the system exceed the minimum required. There is more to fault tolerant design than merely introducing redundant computational units in a haphazard way. Duplicating all computational resources in a system assures fault tolerance at great computational expense. On the other hand, a system in which only a few computational units are duplicated is not fault tolerant with respect to potential failures

[1] We use the phrase 'neural networks' strictly in the sense of artificial neural computing systems, and not biological mechanisms.

in other units. Partial redundancy is only a precondition, and does not ensure robustness. The design of a system may make it selectively fault tolerant to errors in some components but not in others. The challenge in fault tolerant design comes from the goal to use as few redundant resources as possible while ensuring the maximum fault tolerance with respect to as many potential failures throughout the system as possible.

1.2 Neural Networks

The design of neural networks is sometimes motivated by fault tolerant neurophysiological mechanisms in animals. In this context, the distributed nature of human memory has been a well-studied phenomenon: cells in the human brain are constantly dying and new ones taking their place, without significantly affecting memories of past events. Destruction of a small part of a distributed memory does not eliminate any integral piece of information from the memory, although performance as a whole degrades (gracefully); holograms provide a useful analogy. If knowledge representation in neural networks is truly distributed, we should expect similar fault tolerant behavior.

Neural networks often contain a large number of computational elements and links. Hence, in any hardware implementation of a neural network, there is a high probability of some node or link being faulty. Many researchers have assumed that neural networks are fault tolerant; the reasoning is that since there are a large number of computational nodes, some degree of redundancy must exist automatically. However, in applications which require high reliability, the usefulness of a neural network cannot be taken for granted without a clear analysis of its fault tolerance, and without techniques which ensure a desired degree of fault tolerance. This appears to be a relatively new topic of research. Our investigations indicate that very few significant results in this direction have been achieved for commonly used neural network models of non-trivial size, although the importance of this problem has been recognized (see, *e.g.*, recent papers [7, 20, 21, 24]). We have therefore developed methods for measuring the robustness of neural networks designed to solve specific problems, and techniques to ensure the development of neural networks which satisfy well-defined robustness criteria.

In our research, we have analyzed the most commonly used 'feedforward' type of neural networks, trained by back-propagation of error, the method popularized by Rumelhart *et al.* [22]. For convenience of presentation we consider a feedforward neural network with one hidden layer. A neural network with I input nodes, H nodes in the hidden layer, and O nodes in the output layer is conveniently denoted as I-H-O network. A 3-4-2 network is shown below in Figure 1.

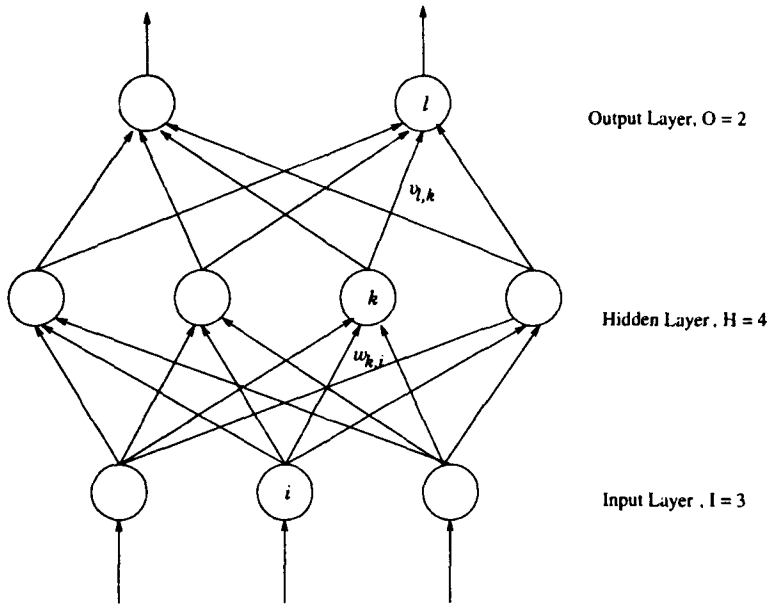


Figure 1: A 3-4-2 neural network.

The computation performed by each hidden and output node is of the form

$$x_j = \frac{1}{1 + e^{-\sum w_{j,i} x_i + \theta_j}}$$

where $w_{j,i}$ is the weight from the i^{th} node of the preceding layer to the j^{th} node, x_i is the output of the i^{th} node, and θ_j is a 'bias' or threshold term that allows the net input to be translated by a desired amount; θ_j will determine the value of x_j when the activation of each node in the preceding layer is $x_i = 0$. The bias is often represented as a weight attached to a special node whose output is always fixed at 1, for notational convenience, and so that the bias can be adjusted by a learning algorithm in the same way as other weights are adjusted. With the inclusion of such a bias node at input and hidden layers, there will be $(I + 1)$ nodes in the input layer and $(H + 1)$ in the hidden layer. There will be $[(I + 1) \times H + (H + 1) \times O] = [(I + O + 1) \times H + O] = K$ links in this neural network . Each node in the hidden layer has $(I + 1)$ fan-in links and O fan-out links.

The problem dictates the size of I and O, the number of input and output nodes, respectively. The number of hidden nodes, H, is chosen in an arbitrary manner. There are two phases in building such a neural network to solve a problem: the training phase followed by the testing phase. In the training phase, the network 'learns' from a set of training samples, *i.e.*, the weights of links and states of computational elements are modified to represent a solution to the desired problem. The performance of the trained network on the test data is then analyzed, in the testing phase.

1.3 Motivation

When we examined oft-touted neural networks published in the literature, we found that even single node/link failures can completely destroy the functionality of the neural network, although it is often claimed that neural networks are fault-tolerant. For instance, consider the encoder-decoder network described in [22] and discussed in various other places. In such a neural network, even the loss of a single node is sufficient to destroy the functionality of the neural network.

This may be an extreme case, since that neural network was optimized to have the smallest possible number of nodes. However, even in neural networks which contain many more nodes than needed from an information-theoretic viewpoint, the redundancy is haphazard, and not well-structured enough to allow us to have any assurance about the robustness of the neural networks. For instance, in a 6-2-1 neural network for solving the symmetry (palindromic strings) problem described in [22], the result of failure of even one node is catastrophic. This happens even if other hidden nodes are introduced and a larger neural network is routinely trained (for the same problem).

All we can conclude from previous work is that it would seem possible to build neural networks in a fault-tolerant way: existing network-building mechanisms do not assure this, and they cannot be relied upon to build fault tolerant neural networks. As hardware implementations of neural networks become widely used in critical applications, research into their fault tolerance and that of fault tolerant neural network design methodology becomes an urgent need.

Faults may occur at either phase, in neural network development as well as in the actual use of the network in an application. Faults occurring in the training phase may slow down the training time, but are less likely to affect the performance of the system. This is because the training process will not be concluded until the faulty components are compensated for by non-faulty parts of the network, assuming that there is enough redundancy and functionality in the neural network. Many physical faults occur at a single component of a neural network, and hence we discuss single faults with greater detail than multiple faults. If faults are detected in the testing phase, retraining with the addition of new resources can solve the problem. Such a fix would not be possible after system development is complete and if faults occur in a neural network application which has already been installed and is in use. Fault tolerant design would ensure the correct functioning of a nonfaulty system in operational use, with graceful degradation in performance if the network's parameters change.

The simplest suggestion to ensure fault tolerance follows explicit redundancy principles, as practiced in classical fault tolerant hardware design, by introducing a multiplicity of elements. In this approach, no implicit assumption is made, but either the entire network

or parts of it are duplicated. In the worst case, each neural network contains several copies of each node and link, with majority estimation being used to resolve errors in the case of faults. This approach is generally not appropriate for neural networks, since no advantage is being taken of the potential for "inherent" fault tolerance in a neural network.

The amount of resources needed by the above approach is painfully large; this can be compensated to some extent by analyzing a neural network using sensitivity parameters (discussed in a later section), and introducing redundant copies for only those nodes and links whose errors are considered critical to the performance of the entire neural network. Even without exhaustive analysis, nodes/links associated with high magnitude values may be duplicated, especially those at the outer layers in a multi-layer neural network.

1.4 Related Work

Belfore [5] has studied performance evaluation of Hopfield-type neural networks with faults. He proposes measures to evaluate the performance of 'average' neural networks, averaging over all possible ways in which a specified number of faults can occur in a neural network. Sequences of neural network state transitions are modeled as Markov chains, and the conditional probability of a neuron firing is obtained using the Boltzmann probability distribution function.

Venkatesh [25] has shown that (in the average case) with random failures, neural networks which implement associative memories are fault-tolerant, with a graceful degradation in storage capacity with increasing losses of interconnections, assuming that each node retains at least about ($\log n$) of its original n interconnections. However, it is not clear that average case analysis suffices for robustness considerations. In particular, the storage capacity of a neural network can be seriously impaired if a large number of connections to one crucial node fail; this is not inconsistent with the results of [25].

If it is possible to detect failures and instantly retrain the network, then graceful degradation of the network capacity is sufficient for fault-tolerance. However, training times are the most expensive aspect of neural computations, and it may not be feasible to halt a system and retrain its neural network component when faults are detected. Hence our concern is with the damage done to existing (already trained) neural networks, rather than with storage capacity alone.

1.5 Overview

In section 2, we present the fault models we have studied. This is followed by a formalization of intuitive notions such as sensitivity and robustness, for failures in neural networks. This enables us to make quantitative judgements in comparing different networks with respect to

their sensitivity. Some theoretical discussions are then presented in section 4. The problems with which we have experimented to judge the performances of suggested methods are described in section 5. We then present the first sets of methods to improve robustness by modifying the learning equations used by neural networks. Section 8 contains the Addition-Deletion algorithm. Hardware-specific issues are discussed in section 9 and in it we suggest an algorithm to improve robustness of neural networks that are to be implemented on hardware that places certain constraints on the magnitudes of weights in the network. This is followed by a comparison of our approach with other competing methods.

2 Fault Models

In digital circuits, commonly occurring faults studied are those resulting when parts of the system are stuck at 1/0 values. Similarly, a connection or link in an analog neural network may be grounded (zero weight value) or saturated (a large weight value). Node failures may also occur, resulting in outputs that are zero or high (saturated) or excessively negative. Our analyses confirm that the location of a node/link failure affects the extent of deterioration of performance of a neural network. For instance, the effect of a fault at the output node is typically higher than the effect of a fault at a hidden node.

Errors in data ("noise") are not considered faults in the neural network; correct performance in the face of noisy data is considered a performance characteristic and may be dependent on the learning algorithm used. Robustness with respect to noise is not the same as being fault tolerant: thus a neural network which handles noisy inputs well may not be fault tolerant, and a fault tolerant system may not handle noisy inputs well for a particular problem.

In our research, we have studied ways of handling the following kinds of single faults in feedforward networks:

1. Severing of an edge in the network, so that the associated weight is treated as zero.
2. Saturation of an edge weight forcing it to be the maximum possible value, if the maximum is constrained to be a finite value.
3. Degradation of a weight value on an edge, reducing its magnitude, and making the value drift towards zero.
4. Random perturbation of a weight value by some percentage of its current value. Note that a perturbation of -100% is equivalent to changing that weight to 0.
5. "Stuck-at-one" failure of a single node, saturating its output to be the maximum magnitude possible.

6. Loss of a node from the network, such that its output is to be treated as zero by the rest of the network.

7. Degradation in the magnitude of a node's output, moving it closer to 0.

8. Random perturbation of a node's output value by some percentage of its current value.

We have also studied the possibility that multiple faults may occur simultaneously in the system. Different fault models are possible when multiple faults can occur in a system. One simple assumption is complete randomness: faults occur independently at different spots at different instances; so the probability of an entire subsystem failure is low, as long as each subsystem is robust to single faults. A more complex model allows for simultaneous failures at different parts of a system.

3 Measures for Evaluating Fault Tolerance of Networks

In order to develop fault tolerant neural computing techniques, the first prerequisite is to have evaluation mechanisms that can measure the fault tolerance of a given neural network. Different evaluation measures may be needed for neural networks intended to perform different tasks such as classification and function approximation. For example, Karnin [13] suggests the use of a sensitivity index to determine the extent to which the performance of a neural network depends on a given node or link in the system. Karnin estimates the sensitivity of each link by keeping track of the incremental changes to the synaptic weights during backpropagating learning. These values are then sorted so that the small value are treated as insensitive links that can be pruned. For a given failure, Carter et al. [7] measure network performance in terms of the error in function approximation; and Stevenson et. al. [23] estimate the probability that an output neuron makes a decision error. Their results are developed for the special case of "Madalines," one class of neural network with discrete inputs.

Fault tolerance and robustness measures can be obtained by theoretical analysis and by experimentation. Since our focus is to improve the robustness of neural networks instead of the analysis of sensitivity, we will measure the robustness using the essential experimental methods. We modify the weight matrix randomly, injecting artificial faults into the system, then measure the change in output values when training inputs are supplied. Broadly speaking, the robustness is measured by comparing output perturbation to a function of the magnitude of the fault. The disadvantage is that it needs to be conducted for large ranges of values and errors, and this will cost much computational time.

In this section, we first design the faulting methods that inject artificial faults to neural networks. Based on these methods and given error measure, we define the sensitivities for

the components of neural networks. To evaluate the fault tolerance of neural networks, we compare the sensitivities of different networks on three classification problems: Fisher's Iris data; four-class grid determination; and five-character recognition. Those results are described in detail in section 5.

3.1 Injecting Artificial Faults

We consider the following possible faults of multi-layered feedforward neural networks, and assume that faults will only occur at hidden nodes and links. The possible link defects are perturbation of weight value, and link stuck at zero, i.e., the value of weight is forced to zero. For node defects, we only consider stuck at 0/1 faults. Specifically, the methods that inject artificial faults to networks are shown below:

1. Single link faults. Perturb one link at a time by changing w_i to $w_i(1+\alpha)$, where $-1 \leq \alpha \leq 1$.
2. Multiple link faults. Randomly inject simultaneous artificial faults to k links.
3. Single link stuck at zero. Force one link weight to remain at zero at a time. Experiments are performed examining worst case and average case, with different links chosen for fault injection.
4. Single node stuck at 0/1. Force one node output to remain at 0 or 1 at a time. The node function used here is sigmoid $1/1+\exp^{-h}$.

Based on fault models presented in the preceding section, we define sensitivities for evaluating fault tolerance of neural networks. Typically, redundancy is obtained by using a large number of nodes in the hidden layer. Consequently, in one-hidden-layer networks, we are interested in developing a measure for the usefulness of only hidden layer nodes. In networks with more than one hidden layer, it is desirable to evaluate the usefulness of various hidden layers. Toward these goals we define link, node, layer, and network sensitivities.

Notation: The vector of all weights (of the trained network) is denoted by $W = (w_1, \ldots, w_K)$. If the i^{th} component of W is modified by a factor α (i.e., w_i is changed to $(1+\alpha)w_i$) and all other components remain fixed, then the new vector of weights is denoted by $W(i,\alpha) = (w_1, \ldots, (1+\alpha)w_i, \ldots, w_K)$. To measure the error of a network, we use mean squared error (MSE),

$$E = \frac{1}{n}\sum_{k=1}^{n}\sum_{j=1}^{O}(o_{kj} - t_{kj})^2,$$

where o_{kj} is the output of an output node for the k^{th} sample and t_{kj} is the target output.

3.2 Link Sensitivity

For a given weight vector W, $E(W)$ denotes the MSE of the network over the training set, and $E_R(W)$ denote the mean square error over the test set R. The effect on MSE of changing W to $W(i,\alpha)$ is measured in terms of the difference

$$s(i,\alpha) = E(W(i,\alpha)) - E(W) \tag{1}$$

or in terms of the partial derivative of MSE with respect to the magnitude of weight change

$$\hat{s}(i,\alpha) = \frac{E(W(i,\alpha)) - E(W)}{|w_i \times \alpha|}. \tag{2}$$

The quantities $s(i,\alpha)$ and $\hat{s}(i,\alpha)$ will be non-negative, because by changing the weight of the trained network we can only decrease its performance, thereby increasing the MSE. This assumes that W represents the result of successful training of the network to minimize MSE, and that the perturbation caused by changing W to $W(i,\alpha)$ does not lead to crossing the energy barrier into the valley containing a different (potentially lower) minimum for the MSE. If $E(W(i,\alpha)) < E(W)$, then a better set of weights must have been accidentally obtained by perturbating W, and retraining can occur for $W(i,\alpha)$. The relative change, α, is allowed to take values from a nonempty finite set A containing values in the range -1 to 1.

Definition 1 Link sensitivity: *Two possible definitions for the sensitivity of the i^{th} link ℓ_i are:*

$$S_\ell(i) = \frac{1}{|A|} \sum_{\alpha \in A} s(i,\alpha) \tag{3}$$

$$\hat{S}_\ell(i) = \frac{1}{|A|} \sum_{\alpha \in A} \hat{s}(i,\alpha).^{2} \tag{4}$$

To compute the sensitivity of each link in a network, all weights of the trained network are frozen except the link that is being perturbed with a fault. $E(W)$ is already known and $E(W(i,\alpha))$ can easily be obtained in one feedforward computation with faulty links.

3.3 Node, Layer, and Network Sensitivities

Node defects may occur in two situations according to our faulting methods. The output of a node may be affected by all the faulted links associated with the node, or caused by the abnormal behavior of the transfer function of the node. Based on these, two definitions are possible for node sensitivity.

[2] $S_\ell(i)$ is an numerical approximation for $\int_{-1}^{1} s(i,\alpha)d\alpha$.

Definition 2 Node sensitivity:

(*i*) *Let $\mathcal{I}_I(j)$ denote the set of all incoming links incident on the j^{th} hidden node, n_j, from the input nodes; let $\mathcal{I}_O(j)$ denote the set of outgoing links from n_j, and let $\mathcal{I}(j) = \mathcal{I}_I(j) \cup \mathcal{I}_O(j)$. The I.-sensitivity (link faulted sensitivity) of a node, n_j, is*

$$S_n^k(j) = \sum_{i \in \mathcal{I}_k(j)} S_\ell(i), \tag{5}$$

where $\mathcal{I}_k(j)$ is the set of the k largest sensitive links in $\mathcal{I}(j)$ and $S_\ell(i)$ is the link sensitivity of link i.

(*ii*) *Let set A be a set of some discrete values between the range of normal node output. For sigmoid function used in usual neural networks, all the values of A will be in the range $(0, 1)$. The N-sensitivity (node faulted sensitivity) of a node, n_j, is*

$$S_n^{nd}(j) = \frac{1}{|A|} \sum_{\alpha \in A} s^\alpha(n_j), \tag{6}$$

where

$$s^\alpha(n_j) = E(W, o(n_j) = \alpha) - E(W),$$

$o(n_j)$ is the output of node n_j, $E(W)$ is MSE of weights W, and $E(W, o(n_j) = \alpha)$ is the MSE with output of node n_j set to be α.

In fault tolerance aspect, we have to consider what is the maximal fault that a system can tolerate. In our problem of the fault tolerance of neural networks, we consider the most critical components of the neural networks. By applying this concept to the definition of node sensitivity, we should consider the most critical link associated to the node. But, when multiple link faults occur in a network, it is possible that a node that was considered most critical with respect to single faults (because its perturbation has the most effect on the network outputs) may not be the most critical with respect to multiple faults. Hence, by merely considering the most critical single link is not sufficient, and the need for a definition of sensitivity, parameterized by the number of faults one may expect in the system. Thus, we formulate the first definition of node sensitivity, examining the most critical links associated with a node.

Since this measure is based on the link faults, it may have some deviation on measuring the node defects, or the abnomal behavior of node transfer function. In this case, we define the second node sensitivity based on node defects. The sensitivity is calculated by summarizing the various deviations of node defects.

Node j	$S_n^1(j)$	$S_n^3(j)$	$S_n^6(j)$	$S_n^{nd}(j)$
0	0.00086	0.00054	0.00041	0.00267
1	0.00431	0.00429	0.00353	0.01340
2	0.00602	0.00542	0.00422	0.01510
3	0.28027	0.21124	0.19246	0.25794
4	0.00063	0.00040	0.00031	0.00233
5	0.09081	0.06033	0.04905	0.14546
6	0.29254	0.35865	0.37223	0.19379
7	0.16116	0.18567	0.20460	0.15304
8	0.02237	0.01769	0.01586	0.05704
9	0.14105	0.15578	0.15732	0.15925

Table 1: Comparison of different measures of sensitivity on Fisher's Iris data with 10 hidden nodes.

Another alternative for the definition of node sensitivity is to combine both definitions defined above to comprise those faulting properties.

Definition 3 Compound node sensitivity: *The C-sensitivity of a node, n_j, is*

$$S_n(j) = p_{lk}(j) \cdot S_n^k(j) + p_{nd}(j) \cdot S_n^{nd}(j), \tag{7}$$

where $p_{lk}(j)$ is the probability that the links associated to node n_j are defected, and $p_{nd}(j)$ is the probability that the output of node n_j is defected.

For convenience of implementation, we assume that $p_{lk}(i)$ is independent to $p_{nd}(i)$, and each node has equally faulted probability.

The comparison of node-defect-sensitivity and link-defect-sensitivity shows that they can both partition the sensitive and insensitive nodes correctly. But the ranks of nodes with close sensitivities are slightly different. For convenience, we normalize the sensitivities of each measure. Table 1 and figure 2 shows the normalized sensitivities of different measures. From figure 2, node 0,1,2,4 and 8 can be classified as insensitivie nodes, while node 3,5,6,7 and 9 as sensitive nodes.

Using these definitions of a node sensitivity, we define layer-sensitivity and the network-sensitivity as follows.

Definition 4 Layer sensitivity: *Sensitivity of layer k is*

$$S_L(k) = \max_{j \in \mathcal{N}(k)} \{\tilde{S}_n(j)\}, \tag{8}$$

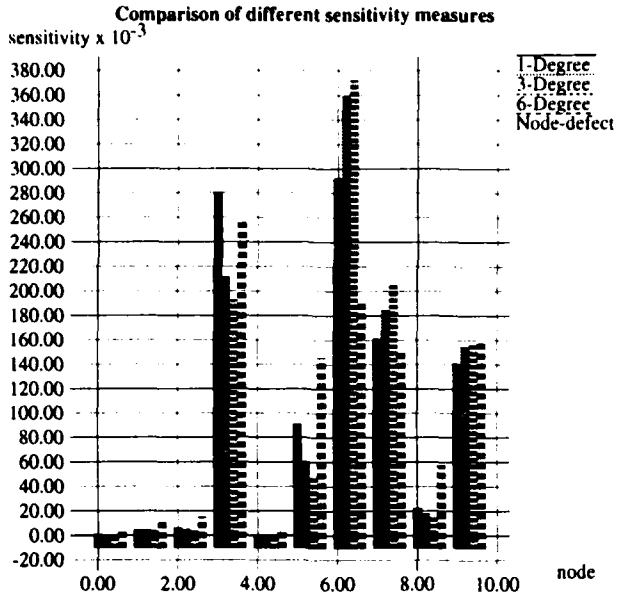


Figure 2: Comparison of different sensitivity measures.

where $\tilde{S}_n(j)$ is one of the node sensitivities defined in definition 2 and $\mathcal{N}(k)$ is the set of nodes at layer k.

Definition 5 Network sensitivity*: Sensitivity of a network N with layers $\mathcal{L}$ is defined as*

$$S_N = \max_{k \in \mathcal{L}} \{S_L(k)\}. \tag{9}$$

For one hidden layer network, it is defined as

$$S_N = \max_{k \in \mathrm{HL(N)}} \{S_n(k)\}, \tag{10}$$

where $HL(N)$ is the set of hidden layer nodes in N.

The fault tolerance of a neural network is measured by injecting artificial faults, introduced in section 3.1, to the network, and then computing the sensitivity of the network using the definitions defined in section 3.1. All the methods (developed in the reported research) to improve the fault tolerance of neural networks are evaluated by comparing the sensitivity of the original network with that of the network evolved using our proposed methods. Robustness of a network is measured in terms of graceful degradation in MSE and MIS(fraction of misclassification errors) on the test set and the training set. We claim a network N_1 is more robust than another network N_2 if N_1 has better performance (i.e. lower MSE or MIS) than N_2 when both the networks are subjected to the same faults.

4 Theoretical Analysis

It is necessary to analyze and understand the behavior of arbitrary neural networks when failures occur. One straightforward approach is to assume that all weights are randomly distributed in a given $I - H - O$ network. We can then estimate the expected change in output value due to the failure of a random node or link.

A more realistic case is to assume that specified nodes are highly correlated (are generally simultaneously ON or simultaneously OFF). We can again estimate the change of output function of the nodes in the next layer when one of these nodes fails. In general, it remains to be explored whether (and to what extent) failures can be compensated simply by modifying the weights of the other (correlated) nodes.

When multiple failures occur in a neural network, it is likely that the worst damage is caused when all failures occur in the same localized region. Robustness of a system must be analyzed with respect to this worst case scenario: what happens when many nodes successively fail in the same physical region (of hardware)? Does performance still degrade gracefully?

The following analysis assumes that a neural network is trained using a method that minimizes mean square error. For simplicity, we assume that the neural network has only one output node and one hidden layer with H nodes. The *MSE* of this network is given by

$$MSE = \frac{1}{P}\sum_{p=1}^{P}(O_p - t_p)^2,$$

where O_p denotes the observed output for the pth pattern and is given by

$$O_p = (1 + e^{-(w_1 y_1 + \ldots + w_h y_H + \theta)})^{-1},$$

and t_p denotes the target output for the pth pattern. In the above expression for O_p, y's denote the outputs of the hidden layer nodes, w's denote the weights associated with links from the hidden layer nodes to the output node, and θ denotes the bias term. In turn, the hidden layer outputs are:

$$y_i = (1 + e^{-(w_{i1} x_1 + \ldots + w_{iI} x_I + \theta_i)})^{-1}.$$

First, we consider the effect of changing the weight w_1 to w_1^0 while all other weights remain fixed. The change in the *MSE* that results from the change in w_1 is derived below.

$$\begin{aligned}
\text{MSE}(w_1^0) - \text{MSE}(w_1) &= \tfrac{1}{P}\sum_{p=1}^{P}\left\{(t_p - O_p^0)^2 - (t_p - O_p)^2\right\} \\
&= \tfrac{1}{P}\sum_{p=1}^{P}\left\{(O_p^0 - O_p)(O_p + O_p^0 - 2t_p)\right\} \\
&= \tfrac{2}{P}\sum_{p=1}^{P}(O_p - t_p)(O_p^0 - O_p) + \tfrac{1}{P}\sum_{p=1}^{P}(O_p - O_p^0)^2.
\end{aligned}$$

The neural network has already been trained, which implies that the difference $(t_p - O_p)$ is negligible for all values of p. Furthermore, since the minimization of MSE implies that

$$\frac{\partial MSE}{\partial w_1} = 0,$$

and

$$\frac{\partial MSE}{\partial w_1} = \frac{\partial\left(\frac{1}{p}\sum_p (t_p - O_p)^2\right)}{\partial w_1} = \frac{2}{P}\sum_p (t_p - O_p)O_p(1 - O_p)y_1,$$

we may conclude that the first term in the previous expression for $\text{MSE}(w_1^0) - \text{MSE}(w_1)$ vanishes, i.e., $\frac{2}{P}\sum_{p=1}^{P}(O_p - t_p)(O_p^0 - O_p) = 0$, whereas the second component of the above expression is positive.

In other words, we would *always* expect MSE to change for the worse, and the above expression allows us to evaluate the amount of deterioration in MSE due to this change. To evaluate this expression, we first consider the difference $(O_p^0 - O_p)$ in the output due to change in weight w_1. To simplify the presentation, we denote $w_2 y_2 + \cdots + w_H y_H + \theta = \alpha$. Then:

$$\begin{aligned}
(O_p^0 - O_p) &= \left[1 + e^{-(w_1^0 y_1 + \alpha)}\right]^{-1} - \left[1 + e^{-(w_1 y_1 + \alpha)}\right]^{-1} \\
&= \frac{e^{-(w_1 y_1 + \alpha)}}{\left\{1 + e^{-(w_1 y_1 + \alpha)}\right\}^2}\left\{1 - e^{-(w_1^0 - w_1)y_1}\right\} \times \left\{\frac{1 + e^{-(w_1 y_1 + \alpha)}}{1 + e^{-(w_1^0 y_1 + \alpha)}}\right\} \\
&= O_p(1 - O_p)\left\{1 - e^{-(w_1^0 - w_1)y_1}\right\}\left\{\frac{1}{1 - (1 - O_p)\left\{1 - e^{-(w_1^0 - w_1)y_1}\right\}}\right\} \\
&= O_p(1 - O_p)\,\epsilon\left\{1 - (1 - O_p)\,\epsilon\right\}^{-1},
\end{aligned}$$

where $\epsilon = 1 - e^{-(w_1^0 - w_1)y_1}$.

Consequently,

$$\text{MSE}(w_1^0) - \text{MSE}(w_1) \approx \frac{1}{P}\sum_{p=1}^{P} O_p^2(1 - O_p)^2\,\epsilon^2\left\{1 - (1 - O_p)\,\epsilon\right\}^{-2}.$$

If we assume that $(w_1^0 - w_1) = \Delta w_1$ is small and further assume that the product $(1 - O_p)\Delta w_1$ is negligible, then to this level of approximation,

$$\text{MSE}(w_1^0) - \text{MSE}(w_1) \approx \frac{(\Delta w_1)^2}{P} \sum_{p=1}^{P} O_p^2 (1 - O_p)^2 y_{1p}^2,$$

since $\epsilon = 1 - e^{-(w_1^0 - w_1)y_1} \approx y_1.\Delta w_1$. This quantity can be determined easily except for the amount of intended change.

Similar results can be obtained when all or some of the weights are changed in a neural network. Our empirical studies confirm the above results, and show how the MSE will deteriorate for large changes in weights.

5 Problems used for experimentation

Three well-known classification problems were used to evaluate the fault tolerance of neural networks in this research. They are described as follows:

Fisher's Iris data

This is a three-class classification problem, in which each sample is a four-dimensional input vector. In building the neural networks, we rescaled the input data to fall between 0 and 1. There are 50 exemplars for each class. In our experiments we obtained a training set of size 100 consisting of the first 34, 35, and 31 exemplars of the three classes, respectively, and saved the remaining 50 exemplars to form the test set.

Four-class grid discrimination problem

In this problem we classify a given two-dimensional observation as belonging to one of four classes. The training sample consisted of 400 exemplars, randomly and uniformly generated to fall into 4 classes. The test set consisted of 600 more similarly generated patterns. The 4-classes and associated input vectors are as shown in Figure 3.

Five-character recognition problem

The five letters "A" to "E" were selected from Letter Image Recognition Data created by David J. Slate. The original 26 letters data was used by Slate to investigate the ability of several variations of Holland-style adaptive classifier systems to learn to correctly guess the letter categories associated with vectors of 16 integer attributes extracted from raster scan images of the letters. The best accuracy obtained was a little over 80%. The objective is to identify each of a large number of black-and-white rectangular pixel displays as one of the 26 capital letters in the English alphabet. The character images were based on 20 different fonts and each letter within these 20 fonts was randomly

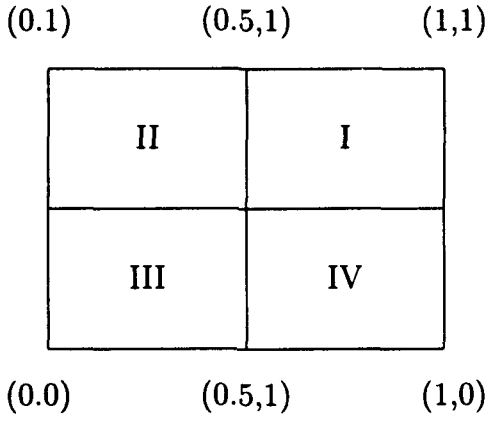


Figure 3: 4-Class Discrimination Problem

distorted to produce a file of 20,000 unique stimuli. Each stimulus was converted into 16 primitive numerical attributes (statistical moments and edge counts) which were then scaled to fit into a range of integer values from 0 through 15.

We select 1000 instances out of 3864 instances as training set, and leave the others as test set. There are 789 instances for "A", 766 instances for "B", 736 instances for "C", 805 instances for "D" and 768 instances for "E". We shuffled all data, then selected the first 1000 instances as training set which including 199 instances for "A", 215 for "B", 194 for "C", 198 for "D" and 194 for "E". The remaining 2864 instances are left as test set.

5.1 Examples of Experimental Results

To show the results of single-link perturbation and multiple-link perturbation, two measures are employed; the average case and the worst case. In the average cases of the single-link perturbation, we plot the sets AC_{mse} and AC_{mis}, whereas in the worst cases we plot the sets WC_{mse} and WC_{mis}, which are defined as follows, where $\mathcal{I}$ is the set of all links, $E_R(W)$ is the mean squared errors and $C_R(W)$ is the fraction of misclassification errors.

- $AC_{mse} = \{(x,y)| -100 \leq x \leq 100, x \bmod 5 \equiv 0, y = \frac{1}{|\mathcal{I}|}\sum_{i\in\mathcal{I}} E_R(W(i, \frac{x}{100}))\}$,
- $WC_{mse} = \{(x,y)| -100 \leq x \leq 100, x \bmod 5 \equiv 0, y = \max_{i\in\mathcal{I}} E_R(W(i, \frac{x}{100}))\}$,
- $AC_{mis} = \{(x,y)| -100 \leq x \leq 100, x \bmod 5 \equiv 0, y = \frac{1}{|\mathcal{I}|}\sum_{i\in\mathcal{I}} C_R(W(i, \frac{x}{100}))\}$,
- $WC_{mis} = \{(x,y)| -100 \leq x \leq 100, x \bmod 5 \equiv 0, y = \max_{i\in\mathcal{I}} C_R(W(i, \frac{x}{100}))\}$.

Figure 4 shows the single-link perturbation of a 4-5-3 neural network after training using backpropagation on Fisher's Iris data.

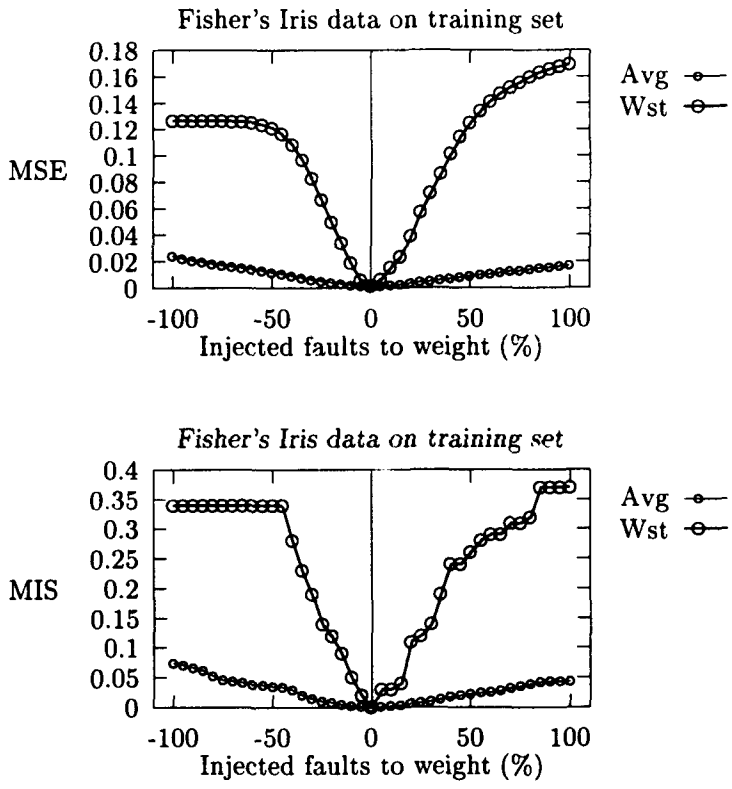


Figure 4: Degradation in MSE and MIS for the training set on single-link perturbation, using networks with 5 hidden nodes, trained on Fisher's Iris data.

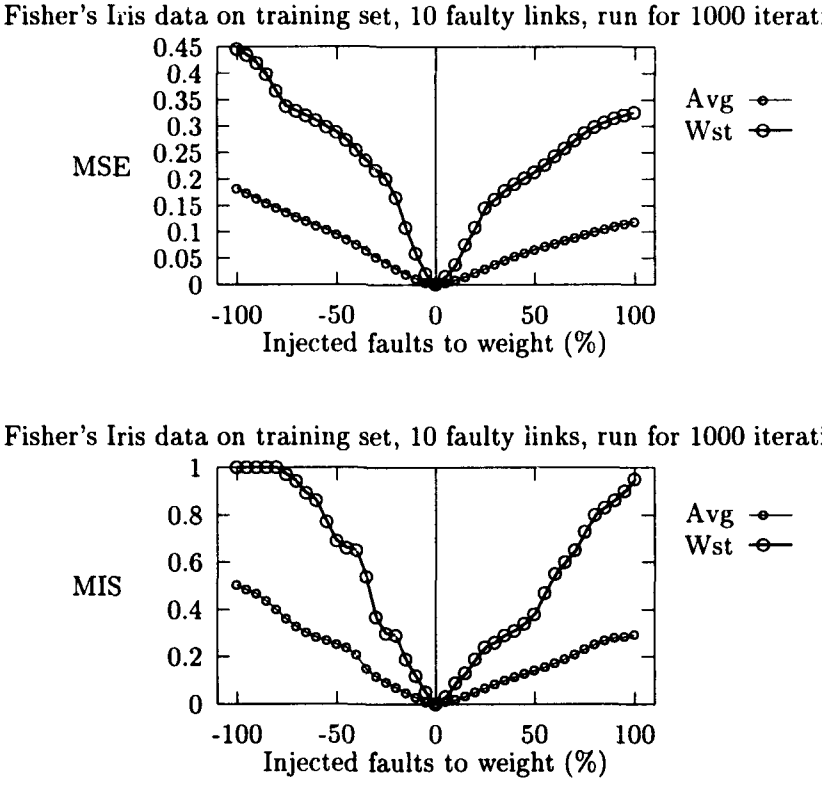


Figure 5: Degradation in MSE and MIS for the training set on multiple-link perturbation, using networks with 5 hidden nodes, trained on Fisher's Iris data. The experiment was run for 1000 times with 10 faulty links selected for each iteration.

To evaluate the robustness of a network with multiple-link perturbation, we randomly selected k links from the network, then injected artificial faults to these links and computed the MSE and MIS of the faulted network. This process was executed for a large number of iterations, then the average case and worst case were plotted out according to these results. Figure 5 shows the example graphs on the multiple-link perturbation of the same network configuration as figure 4. The experiment randomly selected 10 faulty links for 1000 iterations.

6 Performance degradation and network structure

One of the important questions we have addressed in our work is the following: Which of the different nodes and weights in a neural network are crucial in the sense that small errors in them lead to significant changes in the output? For the purpose of analysis, we examined neural networks of 'minimal' size needed to accomplish a given task. Feedforward neural networks with the smallest possible number of hidden nodes, all in one hidden layer, were

examined. Our findings are as follows.

1. In neural networks in which the dimensionality of the input vectors is significantly larger than the number of output nodes, the neural network is more fault tolerant to errors in the weights between the output nodes and the hidden layer nodes. An example problem of this category is two-class classification of samples in an n-dimensional sample space, e.g., diagnostic problems in which many potential symptoms must be analyzed to decide whether a patient has a particular disease.

2. In neural networks in which the dimensionality of the input vectors is significantly smaller than the number of output nodes, the neural network is more fault tolerant to errors in the weights between the input nodes and the hidden layer nodes. An example problem of this category is multiclass classification of samples in two-dimensional input space, e.g., character recognition over the roman alphabet.

3. In intermediate neural networks which do not fall into either of the above categories, all we can say is that the sensitivity of network outputs to different weight values is not uniform. In the problems that we have studied, one or two nodes and weights can most often be isolated such that their influence on the network outputs is significantly higher than the influence of other nodes and weights.

The last-mentioned issue was pursued further by experiments of the following kind. Neural networks of different sizes (with differing numbers of hidden nodes) were trained using the same training algorithm (back-propagation) for the same classification problem. After successful training, faults of various magnitudes were injected into these networks, and the performances of the networks were measured using two criteria:

1. the number of misclassification errors, and

2. the mean square errors, measuring how much the neural network outputs differed from desired outputs.

To ensure that the random choice of the initial weights does not excessively influence the outcome of the experiments, many such experiments were conducted. Average values of the observed misclassification and mean square errors were measured, averaging over the results obtained by injecting faults at different locations in each neural network. We also measured the worst case for the misclassification and mean square errors, i.e., the highest error values obtained when a weight in an neural network was perturbed.

Significant performance degradation results when such faults are injected into a network trained for a four-class classification problem in two-dimensional input space. Each sample belongs to one of four classes. For example, if the two-dimensional input $(x_1, x_2) \in \{(0.5, 1) \times$

$[0.0, 0.5]\}$ then it belongs to class II. The training set T consists of 1000 observations, 250 from each class. For each class, the inputs are generated randomly and associated with the target output vectors given by

$$t = \begin{cases} (0.9, 0.1, 0.1, 0.1) \text{ if the observation is from class I} \\ (0.1, 0.9, 0.1, 0.1) \text{ if the observation is from class II} \\ (0.1, 0.1, 0.9, 0.1) \text{ if the observation is from class III} \\ (0.1, 0.1, 0.1, 0.9) \text{ if the observation is from class IV} \end{cases}$$

Many different neural networks were trained to classify the data in four classes. All the networks have one hidden layer, and contain 2 input nodes and 4 output nodes. These networks are conveniently denoted as 2–h–4 networks, where h is the number of hidden nodes. Neural networks with $h \geq 2$ are successful in classifying the patterns into their respective classes, due to the nature of the problem.

The neural networks were trained with the given training set of 1000 patterns. After training is complete, the weights connecting the input layer to the hidden layer and the hidden layer to the output layer are changed in magnitude.

At least two hidden nodes were necessary for neural network training (using backpropagation) to converge. In the average cases, fault tolerance of the neural networks improves when additional (redundant) hidden nodes are introduced. In other words, for a given percentage perturbation in a weight of the trained network, the average amount of output errors introduced diminishes when we have more than two hidden nodes. This improvement is observed when the number of hidden nodes increases from 2 to 3, and further from 3 to 4. However, the additional improvement is very small when the number of hidden nodes is increased beyond 4.

While these results appeared to coincide with the expectation that fault tolerance improves with the addition of redundant nodes in a neural network, this was contradicted by examination of changes in the worst case errors when additional hidden nodes were introduced. To wit, no improvement in worst case performance was observed by training a neural network with more hidden nodes.

We then re-examined the average case results, and concluded that the apparent improvement in performance (when more hidden nodes were used) was merely a result of averaging over a larger number of nodes and weights. The additional hidden nodes were completely and uselessly redundant: their presence was not making any significant difference to the performance of the neural network. In other words, *backpropagation does not make good use of available redundancy, and fault tolerance is not improved as a result of training a network with more hidden nodes than is absolutely necessary.* These observations are correct for the neural networks that we have examined, and we expect that the same observations will hold for other neural networks.

7 Training to Discourage Large Weights

We now explore how neural learning techniques may be modified for fault tolerance. Most neural learning algorithms are crucially dependent on an error or energy function which is minimized during the training period. It is possible to modify this error function in such a way that the algorithm results in building fault tolerant neural networks.

One method we have examined is the following: a robustness term is introduced into the error function being minimized; this term increases with the partial derivatives $\partial f/\partial g_i$, where f is the neural network output function and g_i is the node output function for a specific (i^{th}) node. The weight given to this term is an additional parameter, *a la* the momentum coefficient.

We now examine an alternate way of improving the fault tolerance of a feedforward neural network. The amounts of perturbation introduced into a network have been specified ($\in A$) as fractions or multiples of the original values, not as fixed quantities. Therefore, network performance is much more sensitive to degradations in large weights than small weights. One possible approach to improve robustness is to build into the training algorithm a mechanism to discourage large weights, but not rule them out altogether (because some problems require networks with large weights). Instead of the mean square error E_0, the new quantity to be minimized is of the form E_0+ (a penalty term monotonically increasing with the size of weights). This penalty term can be chosen in several different ways, and we have implemented three different possibilities; the alternative learning rules minimize error functions E_1[3], E_2[4] and E_3[5], modifying each weight w_i by $-\eta\frac{\partial E_1}{\partial w_i}$, $-\eta\frac{\partial E_2}{\partial w_i}$ and $-\eta\frac{\partial E_3}{\partial w_i}$, where η is the learning rate, and

$$\frac{\partial E_1}{\partial w_i} = \frac{\partial E_0}{\partial w_i} + cw_i, \tag{11}$$

$$\frac{\partial E_2}{\partial w_i} = \frac{\partial E_0}{\partial w_i}(1 + cw_i), \tag{12}$$

$$\frac{\partial E_3}{\partial w_i} = \frac{\partial E_0}{\partial w_i}(1 + \frac{1}{2}c\sum_i w_i^2) + cw_i E_0. \tag{13}$$

Note that the discouragement of high weights in (12) and (13) are less than in (11). Equation (11) amounts to minimization of $E_1 = E_0 + \frac{1}{2}c\sum_i w_i^2$. This and similar cost function has been used earlier, see [11].

Experimental results show that all these modified learning rules do improve the robustness of the network when compared to standard backpropagation, but the improvement is much less than with the robustness-introducing algorithm described later (Figure 8). This

[3] $E_1 = E_0 + \frac{1}{2}c\sum_i w_i^2$.

[4] E_2 is chosen such that the amount by which a high weight is penalized is proportional to the weight as well as the mean square error.

[5] $E_3 = E_0(1 + \frac{1}{2}c\sum_i w_i^2)$.

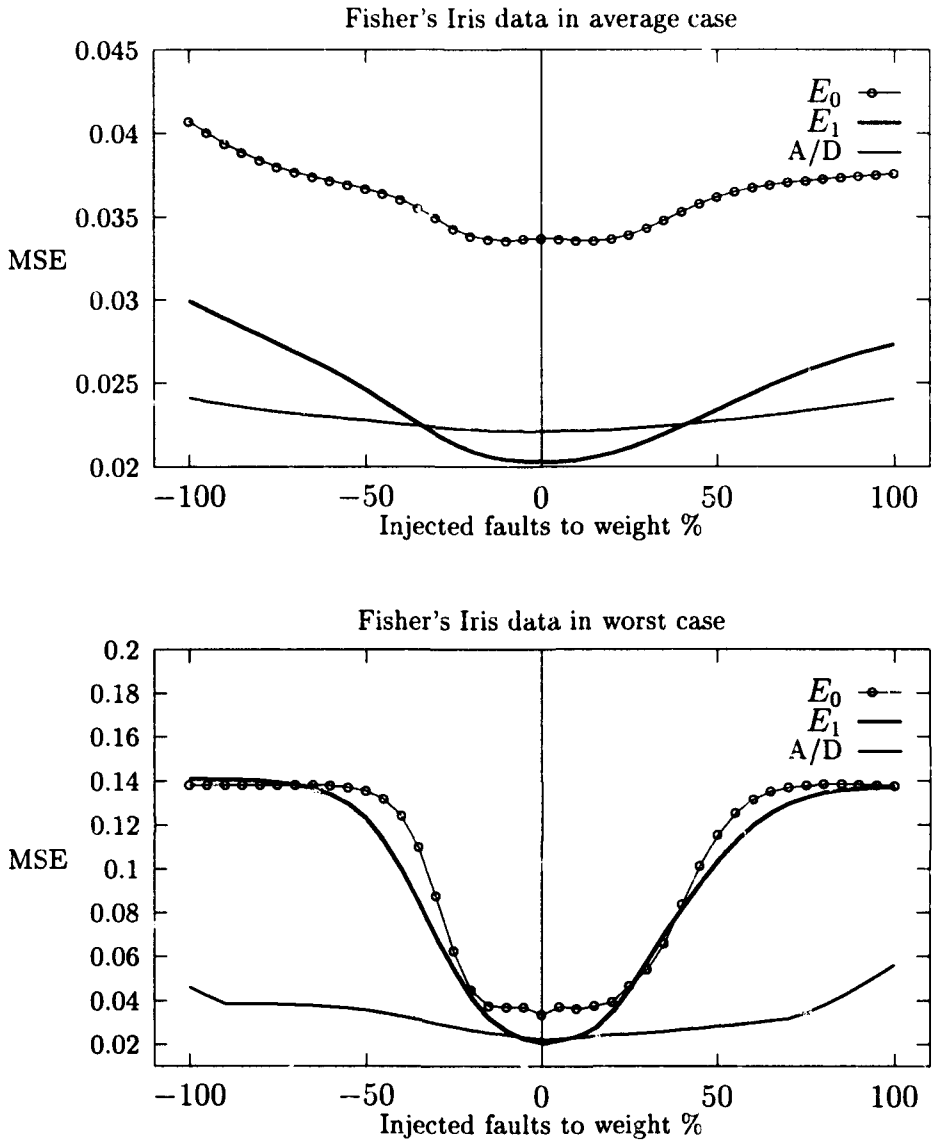


Figure 6: Comparison of traditional backpropagation, alternative learning rule and robustness procedure. These are trained on Fisher's data using Combination 0 with 10 hidden nodes measured in MSE. A/D represents the addition/deletion process using our algorithm. Parameter c is equal to 0.00005.

observation was true for different values for parameter c (in the equations above); Figure 6 and Figure 7 compare the results of this approach, using error measure E_1 and weight change suggested in equation (11), with standard backpropagation and our A/D algorithm described in the next section.

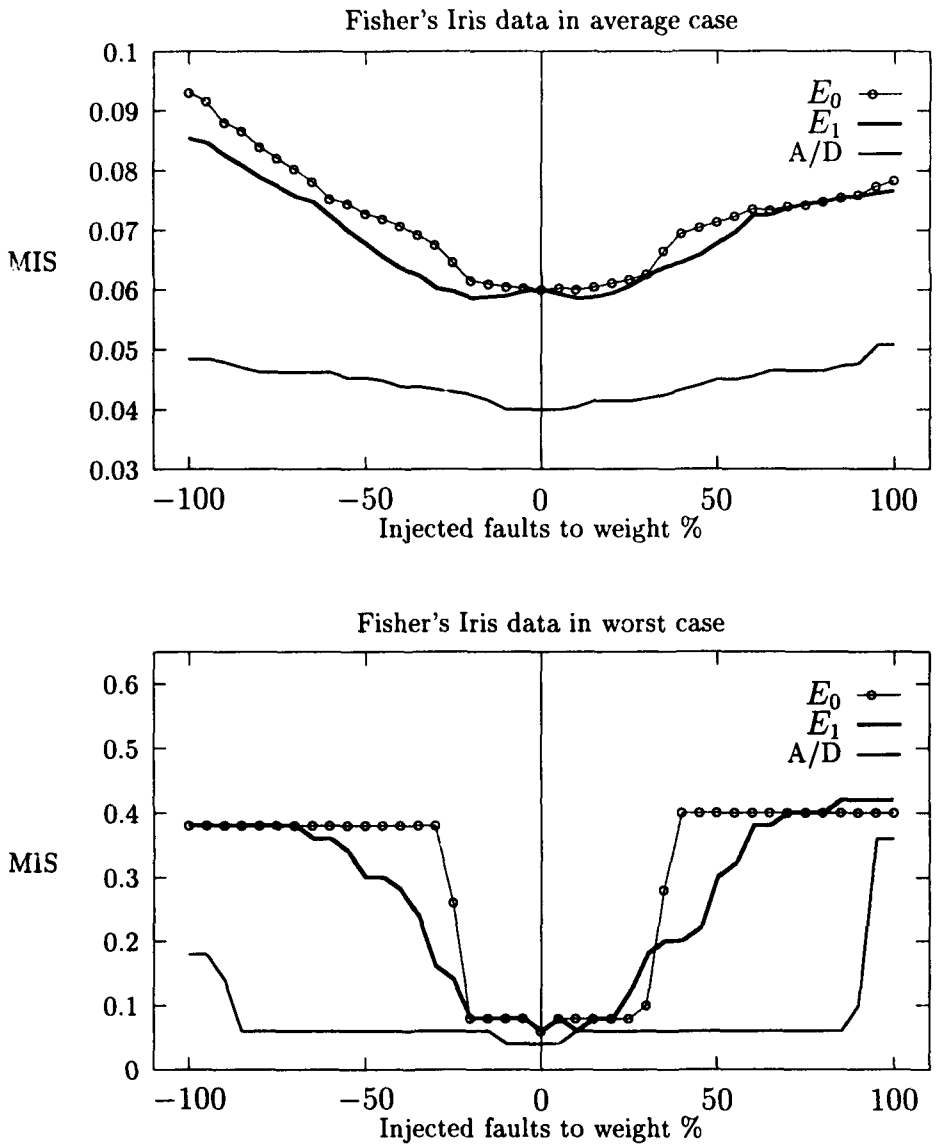


Figure 7: Comparison of traditional backpropagation, alternative learning rule and robustness procedure. These are trained on Fisher's data using Combination 0 with 10 hidden nodes measured in MIS. A/D represents the addition/deletion process using our algorithm. Parameter c is equal to 0.00005.

Let $\mathcal{TR}$ be the training set and $\mathcal{TS}$ be the test set.
Obtain a well-trained weight vector $\mathcal{W}_0$ by training an I-H-O network $\mathcal{N}_0$ on $\mathcal{TR}$.
$i = 0$, and $H^* = H$.
while *terminating-criterion* is unsatisfied **do**
/* $\mathcal{N}_i$ is the i^{th} network, $\mathcal{N}_0$ is the initial network. */
$\varepsilon = S_N(\mathcal{N}_i) \times 0.1$ /* $S_N(\mathcal{N}_i)$ is the worst node sensitivity of the network $\mathcal{N}_i$. */
$\mathcal{N}_{i+1} = \mathcal{N}_i - \{n_j | S_n(n_j) < \varepsilon\}$
$\mathcal{W}_{i+1} = \mathcal{W}_i -$ {all links connected to node n_j}
Retrain the network $\mathcal{N}_{i+1}$.
$H^* = H^* + 1$
$\mathcal{N}_{i+1} = \mathcal{N}_i \cup \{n_{H^*}\}$
$\mathcal{W}_{i+1}$ = Setting the weights of links incident on the new node n_{H^*}, and modifying those connected to the most sensitive node in $\mathcal{N}_i$
$i = i + 1$
end while

Figure 8: Addition/Deletion procedure for improved fault tolerance.

8 Addition/Deletion Procedure

In this section we present a procedure to build robust neural networks against link weight changes. We also discuss several variations of the algorithm to improve robustness, and compare their performance with each other and with an alternative learning rule $R1$ which gives an extra penalty term to the delta rule to prevent weights growing too large. We then discuss the extension of these to networks in which multiple faults occur. Our results show considerable improvement in robustness over randomly initialized networks, trained using the standard backpropagation algorithm, which were of the same size as the networks developed using our algorithm.

Our methodology can be briefly summarized as follows. Given a well-trained network, we first eliminate all "useless" nodes in hidden layer(s). We retrain this reduced network, and then add some redundant nodes to the reduced network in a systematic manner, achieving robustness against changes in weights of links that may occur over a period of time. The process of retraining may transform the network, making some other nodes eliminable, in which case the deletion-addition process is repeated.

8.1 Elimination of Unimportant Nodes

To determine the proper size of a neural network for a specific problem is not easy. In most networks, the number of nodes available exceeds the minimum required to ensure that the network will solve the problem, with the hope that the extra nodes assure enough redundancy to sustain fault tolerance. In practice, however, we have observed that many of these extra nodes serve no useful purpose, and traditional network training algorithms do not ensure that the redundant nodes improve fault tolerance. An analogy is that of building in extra processors into a computer without ensuring that these processors are properly connected to the rest of the components of the machine.

Once a network has been trained, the importance of each hidden layer node can be measured in terms of its sensitivity. Given a reference sensitivity ϵ, node n_j is removed from the hidden layer if $S_n(j) \leq \epsilon$. The value of ϵ can be adjusted such that elimination of all such nodes makes little difference in the performance of the reduced network compared to the original network. In our experiments, described in Section 4, we have used $\epsilon = 10\%$ of the maximum node sensitivity. This relative measure for reference ϵ is the choice that we have found to work well in both experiments, but other choices of ϵ could also work well.

The deletion of nodes with a small sensitivity (the unimportant nodes) results in an I-H^*-O network that should perform almost as well as the original network. We have observed in some experiments that H^* may be considerably smaller than H.

8.2 Retraining of Reduced Network

Removal of unimportant nodes from the network is expected to make little difference in the resulting MSE. But the resulting network with reduced dimensionality of weight space may not be in its (local) minimum, due to the following reason. If $(x_1, ..., x_n)$ is a local minimum of a function $f^{(n)}$ of n arguments, there is no guarantee that $(x_1, ..., x_{n-i})$ is a local minimum of a function $f^{(n-i)}$ defined as $f^{(n-i)}(x_1, ..., x_{n-i}) \equiv f^{(n)}(x_1, ..., x_{n-i}, 0, ..., 0)$. For our problem, $f^{(n)}$ and $f^{(n-i)}$ are the MSE functions over networks of differing sizes, where the smaller one is obtained by eliminating some parameters of the larger network. Retraining of the reduced network will locate the MSE at a (local) minimum in the new weight space. In our experiments we have observed that the number of iterations needed to retrain the network to achieve the previous level of MSE is usually small.

At this stage we have obtained a network that is "well-trained" and devoid of redundant nodes, but it does not satisfy robustness against link faults. In the following section, we describe a criterion to build a robust net out of this "lean" well-trained network.

8.3 Addition of Redundant Nodes

To enhance robustness, our method is to add extra hidden nodes, in such a way that they share the tasks of the critical nodes—nodes with "high" sensitivity.

Let $w_{i,k}$ denote the weight from the k^{th} input node to the i^{th} hidden layer node, and let $v_{i,k}$ denote the weight from the k^{th} hidden node to the i^{th} output node. Let the j^{th} hidden node have the highest sensitivity, in a I-H^*-O network. Let $h = H^*$. Then the new network is obtained by adding an extra $(h+1)^{th}$ hidden node. The duties of the sensitive node are now shared with this new node. This is achieved by setting up the weights on the new node's links as defined by:

(1) First layer of weights: $w_{h+1,i} = w_{j,i}, \forall i \in I$, {the new node has the same output as the j^{th} node}

(2) Second layer of weights: $v_{k,h+1} = \frac{1}{2} v_{k,j}, \forall k \in O$, {sharing the load of the j^{th} node}

(3) Halving the sensitive node's outgoing link weights $v_{k,j}, \forall k \in O$.

In other words, the first condition guarantees that the outputs of hidden layer nodes n_j and n_{H^*+1} are identical, whereas the second condition ensures that the importance of these two nodes is equal, without changing the network outputs.

After adding the node n_{H^*+1}, node sensitivities are re-evaluated and another node, n_{H^*+2}, is added if the sensitivity of a node is found to be 'too' large. On the other hand, a node is removed if its sensitivity is 'too' low. Our primary criteria for sensitivity of a link and a node are equations (3) and (7). In our experiments, we have found that there is not much difference in the results obtained using the other definitions of sensitivity. A node is deleted if its sensitivity is less than 10% of the sensitivity of the most critical node.

We continue to add nodes until the termination criterion is satisfied, i.e., the improvement in the network's robustness is negligible. We have experimented with two termination criteria. The first criterion is adding extra nodes until the sensitivity of the current most critical node is less than some proportion of the sensitivity of the initial most critical node. The second criterion is adding extra nodes until the number of nodes are equal to the original number of nodes, in order to compare two networks of the same size.

Notation: Let $E(N^{\alpha}_{mn})$ denote the error obtained when the link weight ν_{mn} in the network N is replaced by $(1+\alpha)\nu_{mn}$. Similarly, let $E(N^{\alpha}_{k})$ denote the average error obtained when each of the link weights ν_{mk} in the network N is replaced by $(1+\alpha)\nu_{mk}$. We remind that ν_{ij} denotes the weight on the link from the j^{th} hidden node to the i^{th} output node.

Theorem: Let N be a well-trained[6] I-h-O network in which link ν_{ij} is more sensitive than every other link in the second (ν) layer, where sensitivity is defined as the additional error resulting from perturbation of any ν_{mn} to $(1+\alpha)\nu_{mn}$, for some α (i.e., with the singleton perturbation

[6] "Well-trained" means that the network error is almost 0.

set $A = \{\alpha\}$) such that these perturbations degrade performance, (i.e., the error $E(N) < \max_{m,n}\{E(N^{\alpha}_{mn})\}$. Let M be the network obtained by adding a redundant $(h+1)^{st}$ hidden node to M and adjusting weights of links attached to this new node and to the j^{th} hidden node, as specified in the addition/deletion algorithm given earlier. Then M is more robust than N, i.e., the sensitivity of M is lower than that of N.

The above theorem pertained to the special case where A was a singleton set. This result extends to the case when A contains many elements, and for node faults; details are given in the Appendix (for continuity of presentation). As shown in the Appendix, the theorem holds even with minor variations in the definitions of the sensitivity. A premise of the above theorem is that perturbations should degrade performance: if such is not the case, i.e., if network error actually decreases as a result of introducing "faults" into the system, then our algorithm replaces the network by the new 'perturbed' network with better performance, and retrains that network.

8.4 Comparison of Robustness of Different Algorithms

In the following subsections, we compare the robustness of neural network training by traditional backpropagation (BP) learning using the generalized delta rule, the learning rule R_1 (described below), and with variations of our add/delete (A/D) algorithm (discussed above) in which the network is further retrained after adding nodes. A node is deleted if its sensitivity is less than 10% of the sensitivity of the most critical node.

We have used two termination criteria in our experiments. The first criterion is adding extra nodes until the sensitivity of the current most critical node is less than some proportion of the sensitivity of the initial most critical node. The second criterion is adding extra nodes until the number of nodes are equal to the original number of nodes. The second criterion is used to obtain a comparison of two networks of the same size.

The weight modification rule R_1 was defined as follows:

$$\Delta w_i = -\eta(\frac{\partial E_0}{\partial w_i} + cw_i) \tag{14}$$

This rule is used to discourage large weights in the network, improving the robustness of the network because degradations in large weights may affect the network outputs to a large extent.

As usual, robustness of a network is measured in terms of MSE (mean square error) and MIS (fraction of misclassification errors) on the test set and the training set. Again, two measures are employed; the average case and the worst case. In the average cases, we plot the sets AC_{mse} and AC_{mis}, whereas in the worst cases we plot the sets WC_{mse} and WC_{mis}, which are defined as follows,

- $AC_{mse} = \{(x,y)| -100 \leq x \leq 100, x \bmod 5 \equiv 0, y = \frac{1}{|\mathcal{I}|}\sum_{i\in\mathcal{I}} E_R(W(i, \frac{x}{100}))\}$,
- $WC_{mse} = \{(x,y)| -100 \leq x \leq 100, x \bmod 5 \equiv 0, y = \max_{i\in\mathcal{I}} E_R(W(i, \frac{x}{100}))\}$,
- $AC_{mis} = \{(x,y)| -100 \leq x \leq 100, x \bmod 5 \equiv 0, y = \frac{1}{|\mathcal{I}|}\sum_{i\in\mathcal{I}} C_R(W(i, \frac{x}{100}))\}$,
- $WC_{mis} = \{(x,y)| -100 \leq x \leq 100, x \bmod 5 \equiv 0, y = \max_{i\in\mathcal{I}} C_R(W(i, \frac{x}{100}))\}$,

where $\mathcal{I}$ is the set of all links and $C_R(W)$ is the fraction of misclassification errors on test set.

8.5 Experimental Results

We evaluate our algorithm by comparing the sensitivity of the original network (with redundant nodes, randomly initialized and trained using the traditional backpropagation algorithm) with that of the network evolved using our proposed algorithm.

We performed four series of experiments for each problem using the following combinations of sensitivity definitions, where A is the set of values by which a weight is perturbed, when testing sensitivity.

Combination 0: Maximal node sensitivity, and $A = \{-1\}$.

Combination 1: Normalized maximal node sensitivity and $A = \{-1\}$.

Combination 2: Normalized maximal node sensitivity and $A = \{+0.1, -0.1\}$.

Combination 3: Normalized maximal node sensitivity and $A = \{\pm 1, \pm\frac{1}{2}\}$.

The same nodes are found to be most sensitive using each of Combinations 0, 1, and 3, since they all examine large perturbations in weights. Experimental results are shown first for Combination 0, and for Combinations 1, 2 and 3 in the next subsection. We have presented the experimental results on Fisher's Iris data, four-class discrimination on grid and two-character recognition data. Each experiment presents the following results:

1. comparison of training using BP and R_1,
2. comparison of BP with further retraining and without further retraining after adding node, and
3. evaluation of multiple faults based on new measure of sensitivity.

For convenience, we denote a network trained by BP as $\mathcal{N}_{BP}$ and trained by R_1 as $\mathcal{N}_{R_1}$, the network $\mathcal{N}_{rule}$ after running A/D process by $\mathcal{N}_{rule-AD}$, where $rule$ is either BP or R_1. We have compared

o N_{BP} vs. $N_{BP\text{-}AD}$,
o N_{R_1} vs. $N_{R_1\text{-}AD}$,
o N_{BP} vs. $N_{R_1\text{-}AD}$,
o $N_{BP\text{-}AD}$ vs. $N_{R_1\text{-}AD}$.

Experimental results show that our definition of sensitivity and A/D process are effective for multiple faults.

8.5.1 Fisher's Iris Data

This is a three-class classification problem, in which each sample is a four-dimensional input vector. In building the neural networks, we rescaled the input data to fall between 0 and 1. There are 50 exemplars for each class. In our experiments, we obtained a training set of size 100 consisting of the first 34, 35, and 31 exemplars of the three classes, respectively, and saved the remaining 50 exemplars to form the test set.

On Fisher's data, results of the comparison of different learning rules R_1 and BP are shown in Figure 9 (MSE) and Figure 10 (MIS, mis-classification) for training data, and Figure 11 and Figure 12 for test data. To start with, we trained a 4-10-3 neural network. The algorithm shown in Figure 8 reduced it to a 4-4-3 network in the first deletion step and then it was built up, successively, to a 4-10-3 network. The deletion/addition process, represented in the form 10 4 5 6 7 8 9 7 8 9 10, implies that in the original network there were 10 hidden nodes, our criterion reduced it to 4, then increased it to 10 as described in Table 2. When a 9-node network was obtained in this manner and retrained, two nodes could again be removed due to the sensitivity criteria, and more nodes were then added following our algorithm in Figure 8. The original 10-node network was found to be roughly as robust as a 6-node network for high perturbations, and worse than all other cases for small perturbations.

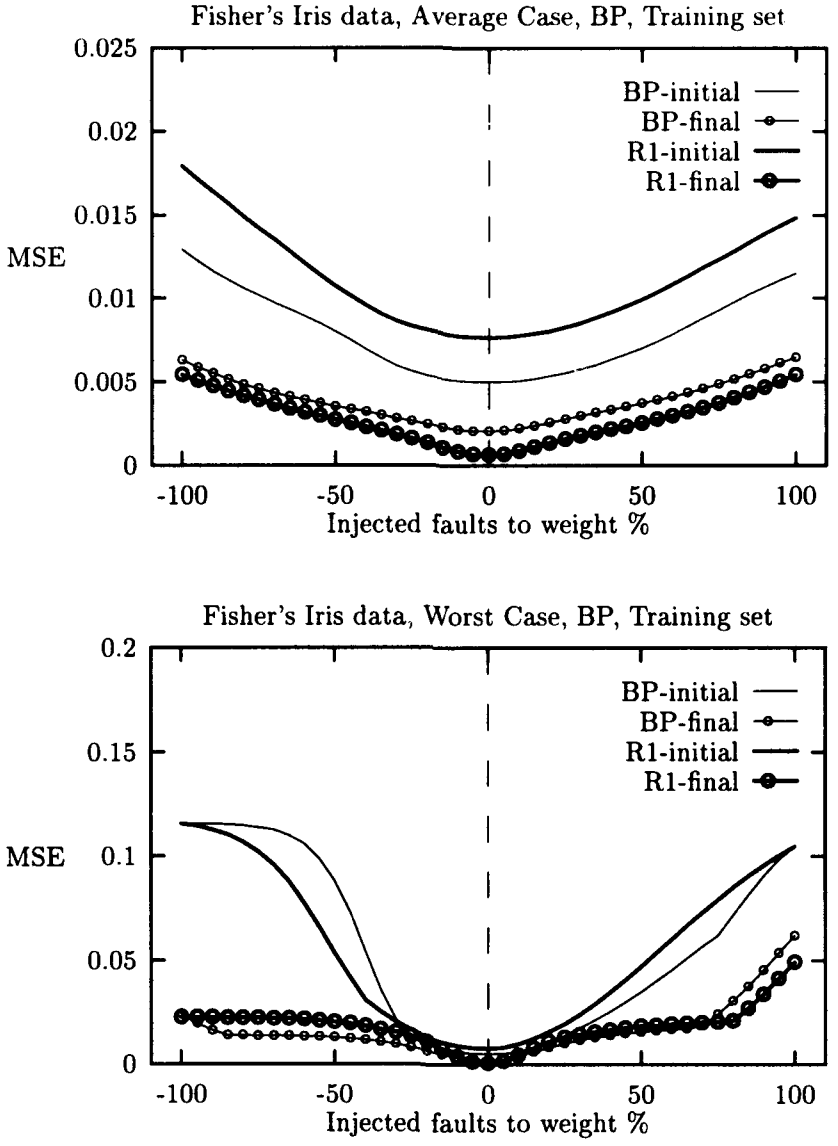


Figure 9: Comparison of BP and $R1$ on the MSE of Fisher's data on training set.

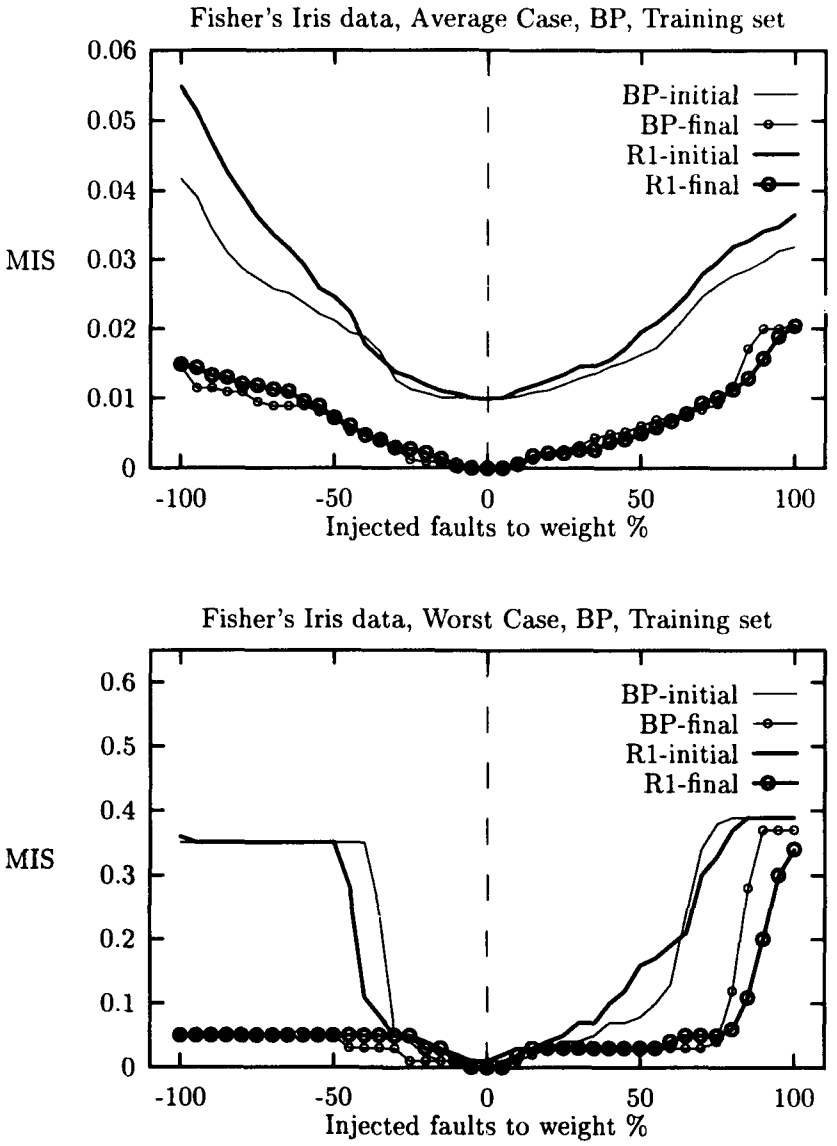


Figure 10: Comparison of BP and $R1$ on the MIS of Fisher's data on training set.

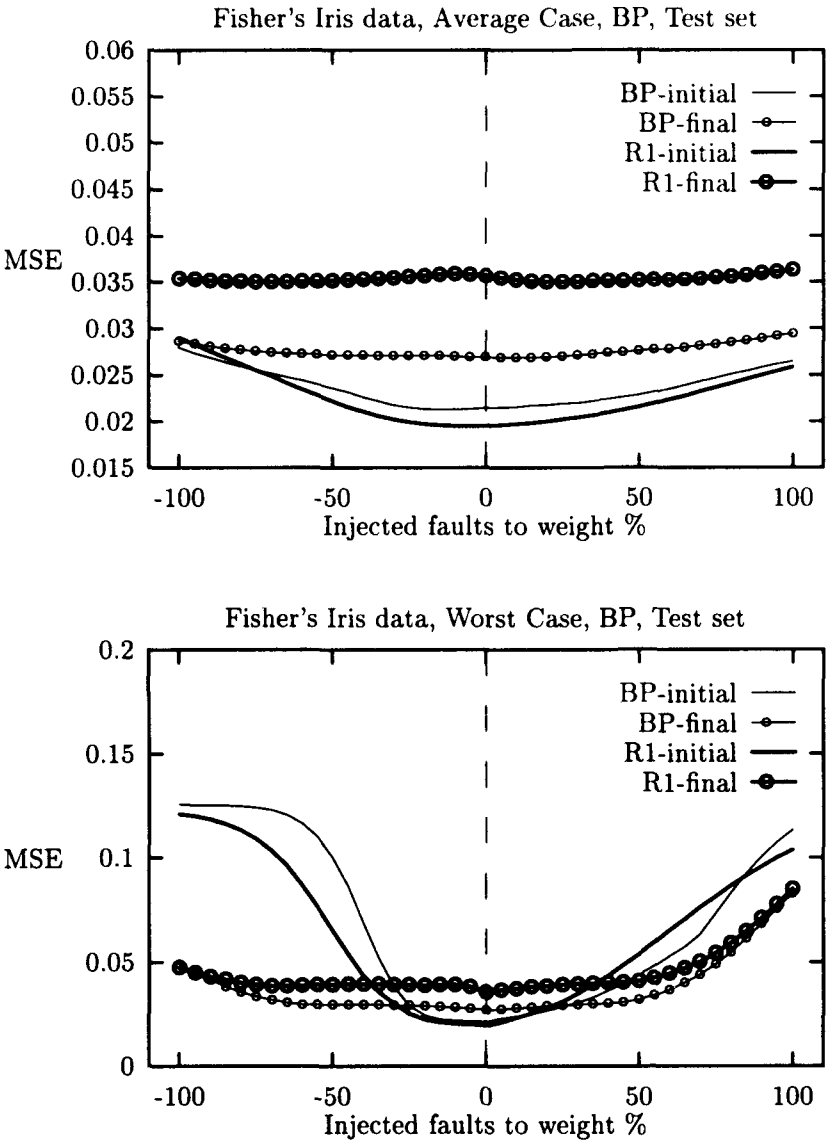


Figure 11: Comparison of BP and $R1$ on the MSE of Fisher's data on test set.

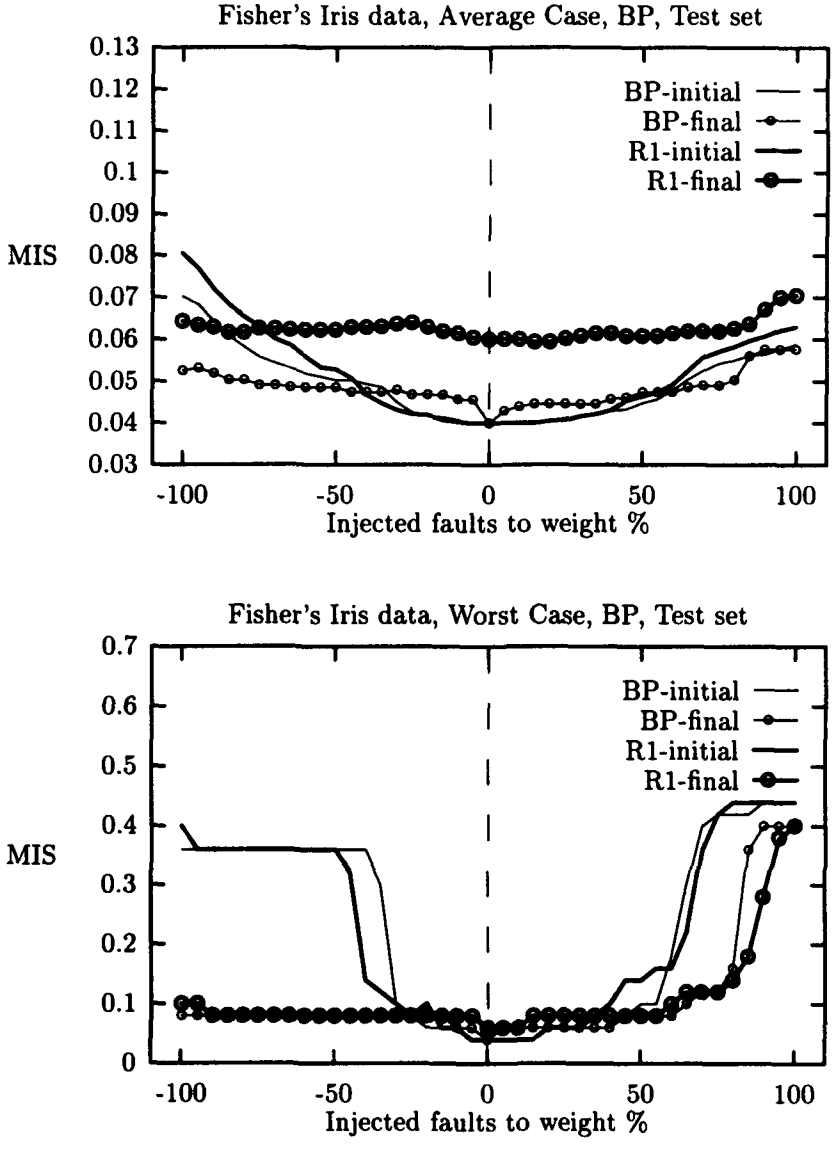


Figure 12: Comparison of BP and $R1$ on the MIS of Fisher's data on test set.

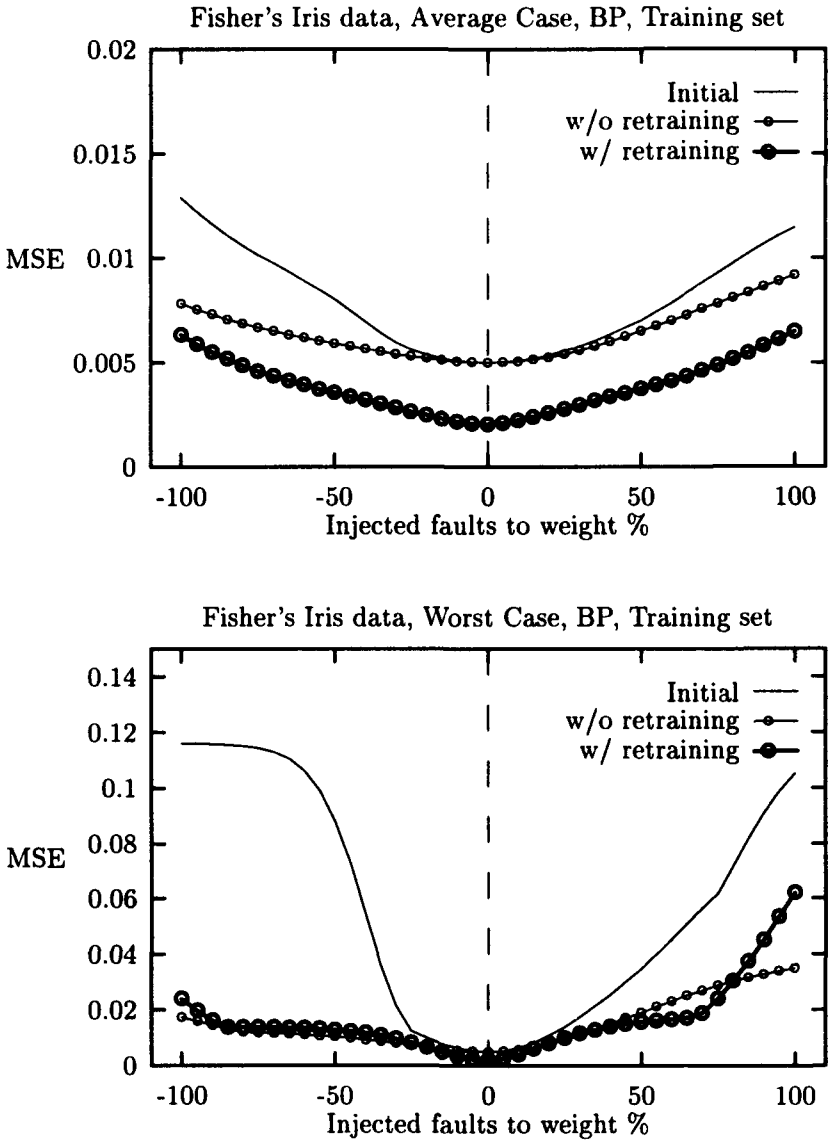


Figure 13: Comparison of *BP* with and without further retraining after adding node on the MSE of Fisher's data on training set.

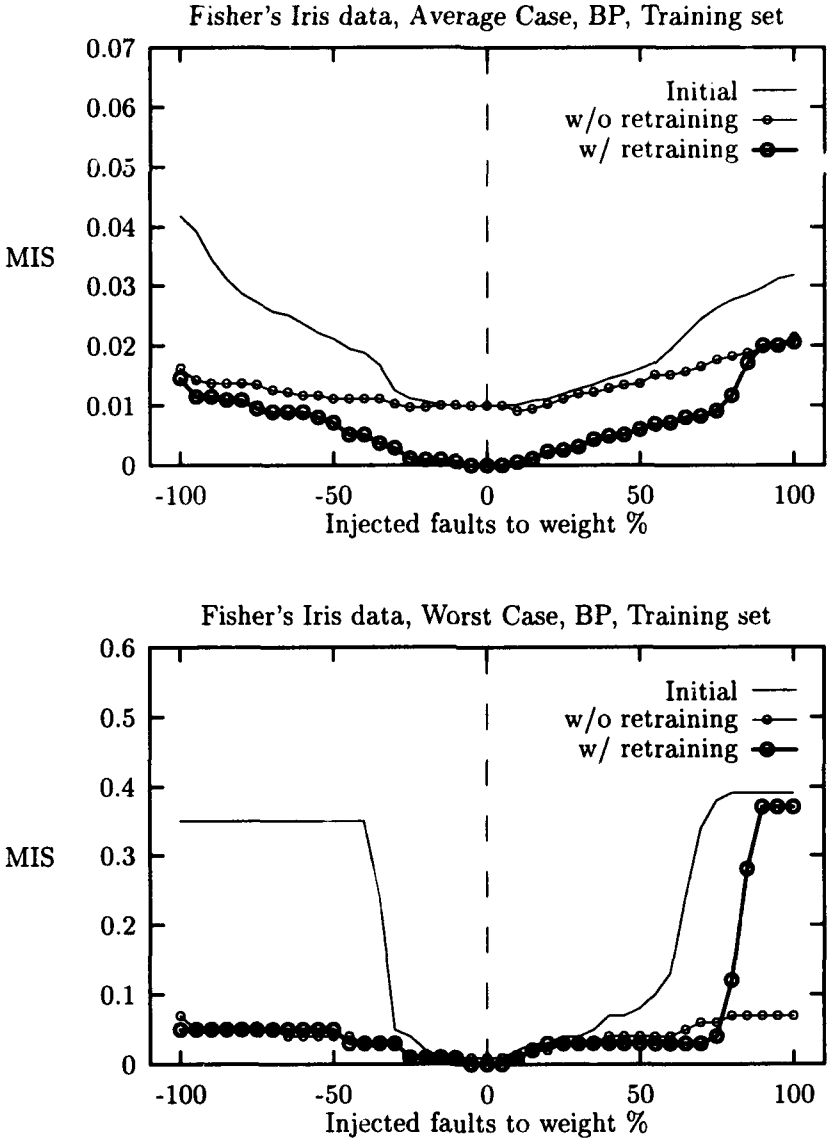


Figure 14: Comparison of *BP* with and without further retraining after adding node on the MIS of Fisher's data on training set.

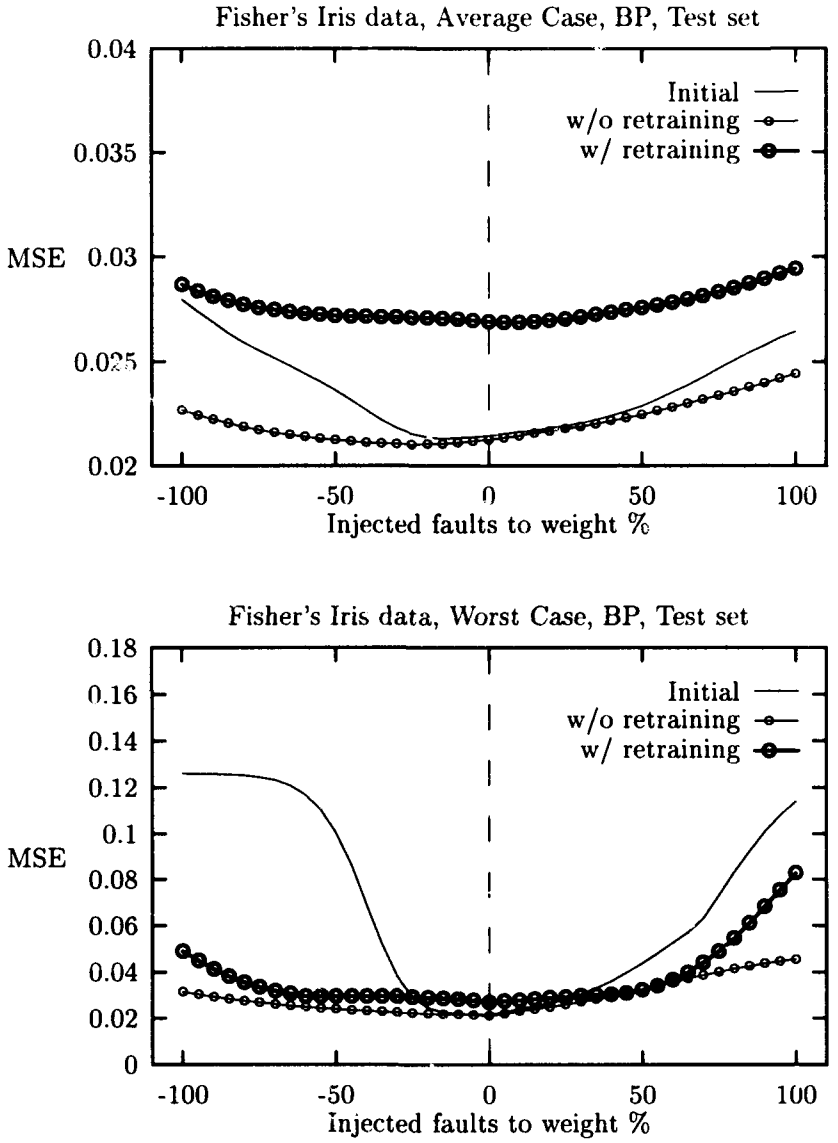


Figure 15: Comparison of BP with and without further retraining after adding node on the MSE of Fisher's data on test set.

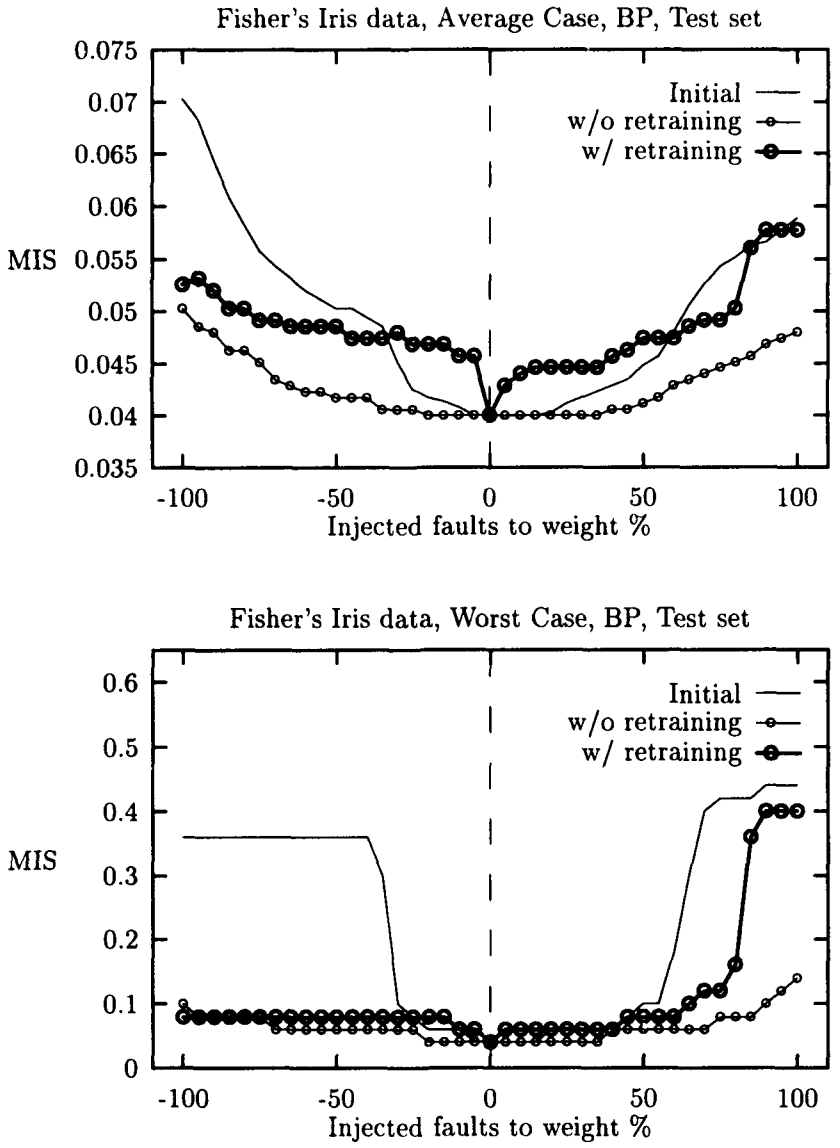


Figure 16: Comparison of *BP* with and without further retraining after adding node on the MIS of Fisher's data on test set.

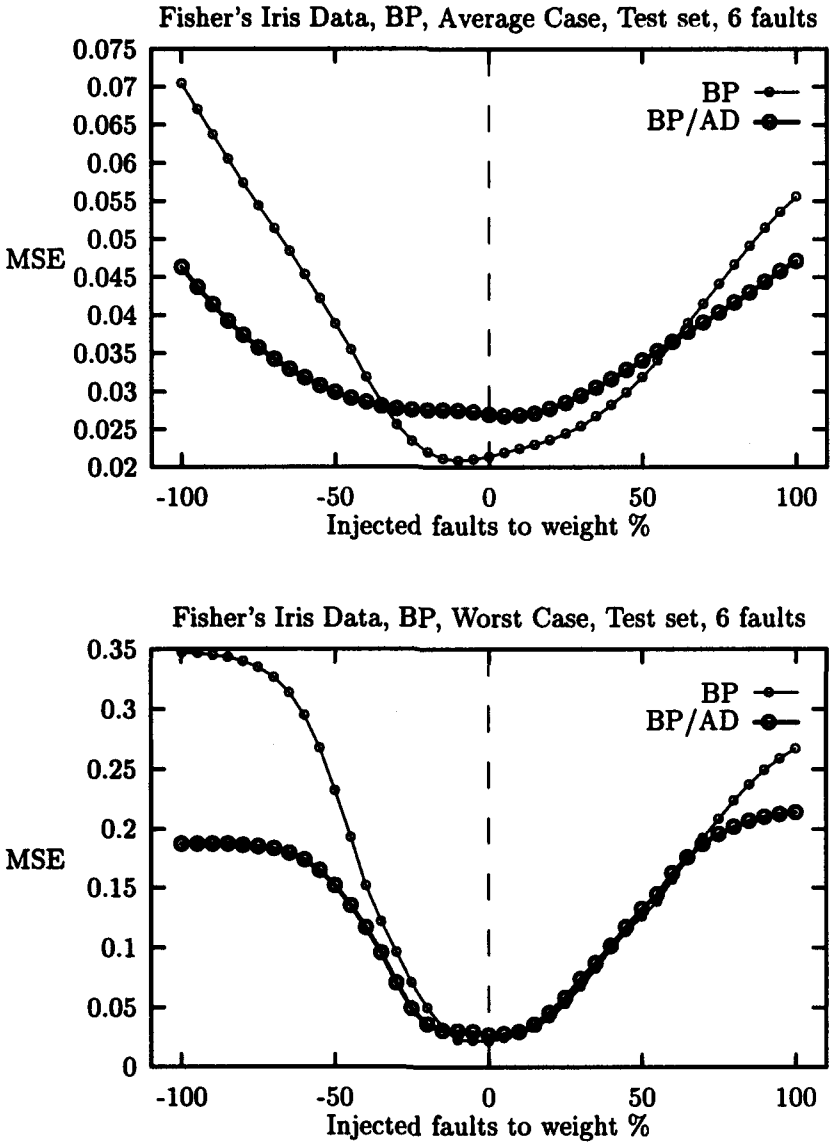


Figure 17: Comparison of $\mathcal{N}_{BP}$ and $\mathcal{N}_{BP-AD}$ on the MSE of Fisher's data on test set using multiple faults evaluation.

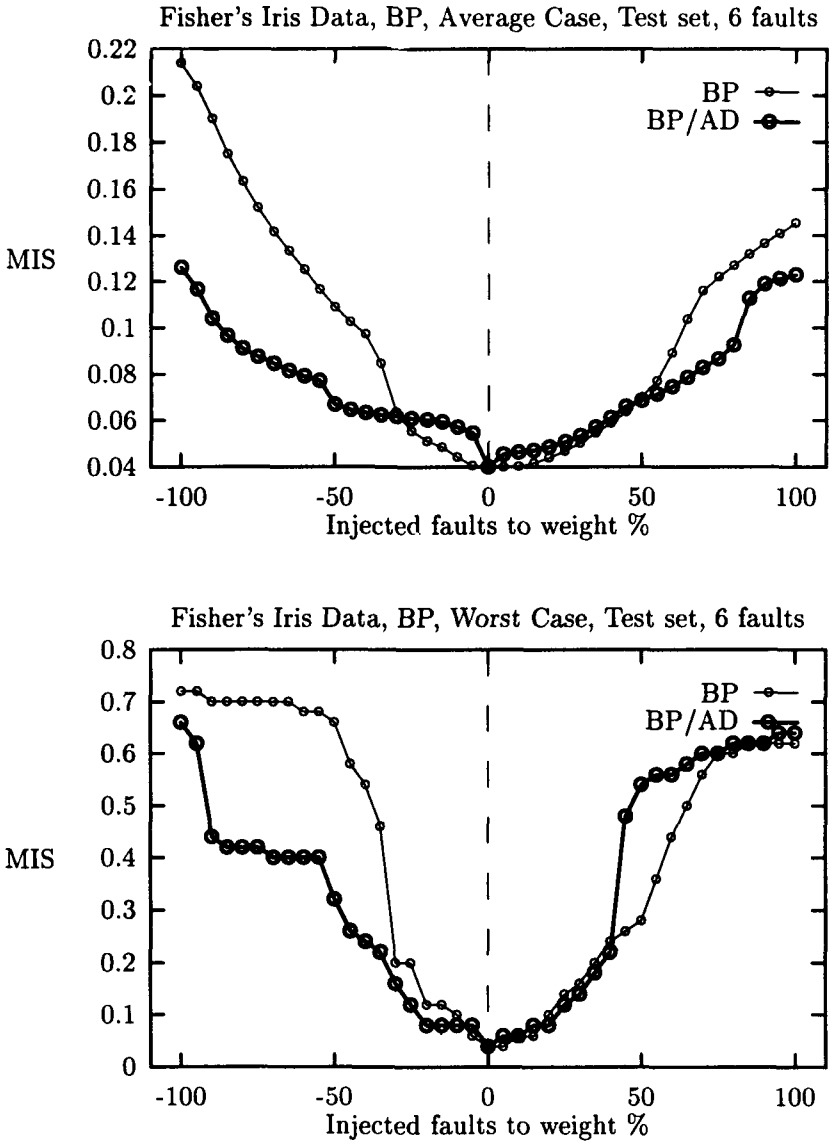


Figure 18: Comparison of $\mathcal{N}_{BP}$ and $\mathcal{N}_{BP-AD}$ on the MIS of Fisher's data on test set using multiple faults evaluation.

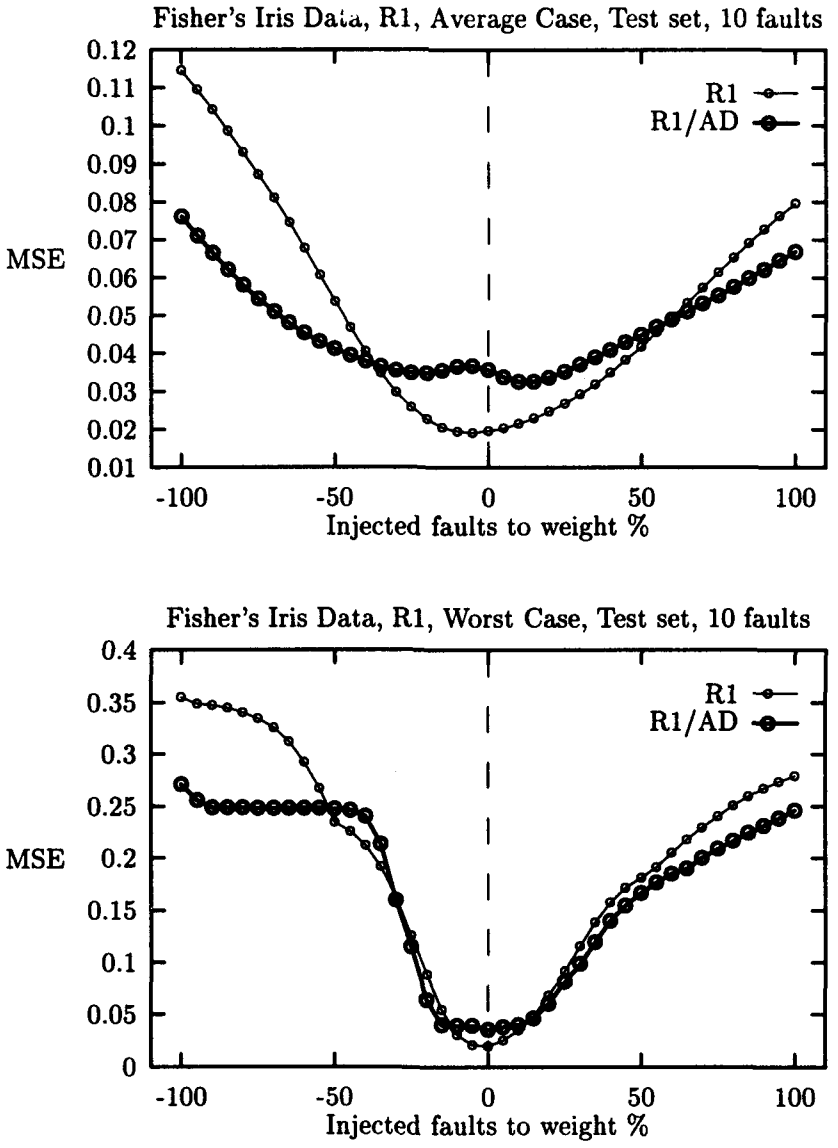


Figure 19: Comparison of $\mathcal{N}_{R_1}$ and $\mathcal{N}_{R_1-AD}$ on the MSE of Fisher's data on test set using multiple faults evaluation.

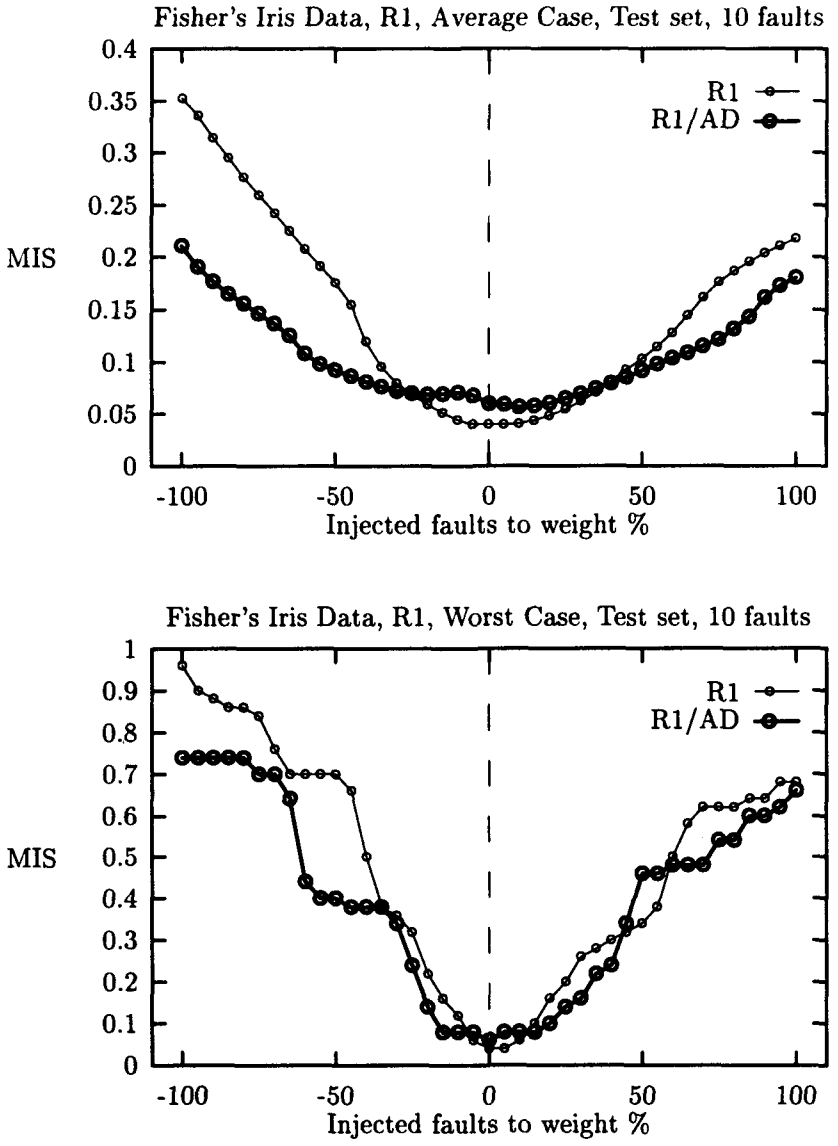


Figure 20: Comparison of $\mathcal{N}_{R_1}$ and $\mathcal{N}_{R_1-AD}$ on the MIS of Fisher's data on test set using multiple faults evaluation.

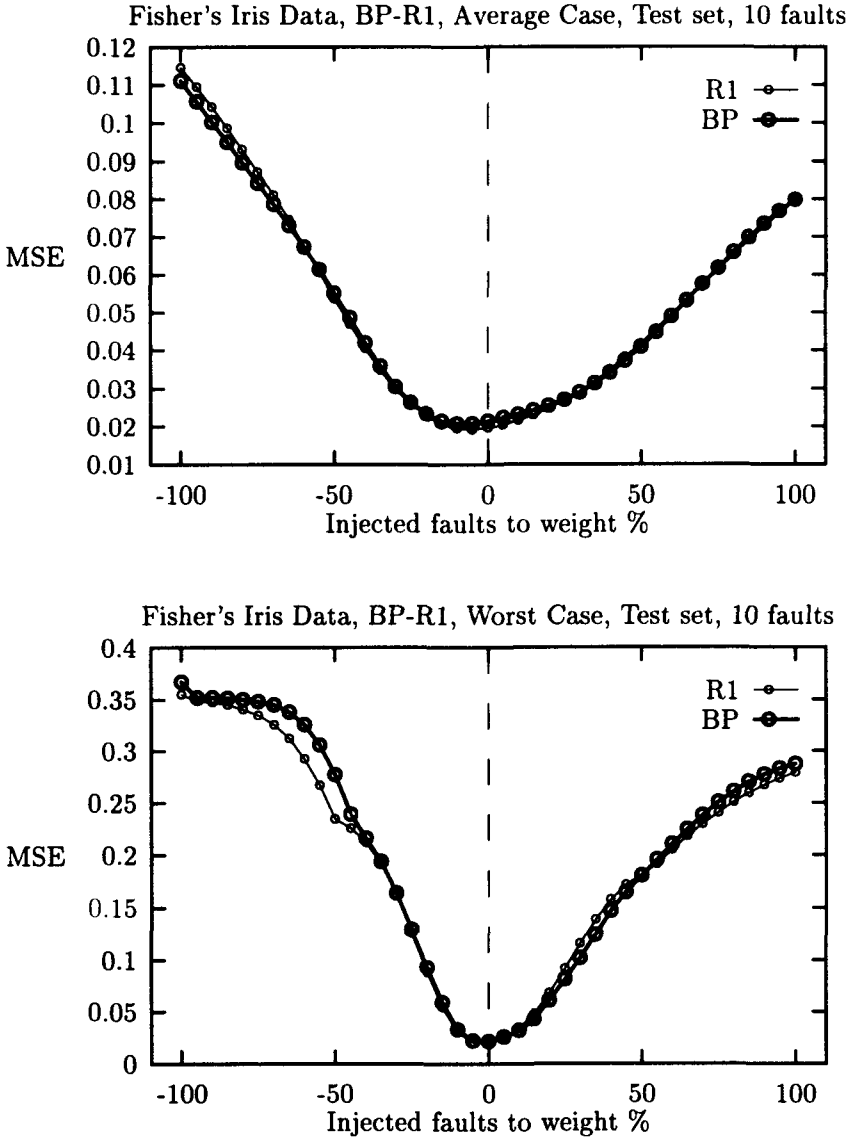


Figure 21: Comparison of $\mathcal{N}_{BP}$ and $\mathcal{N}_{R_1}$ on the MSE of Fisher's data on test set using multiple faults evaluation.

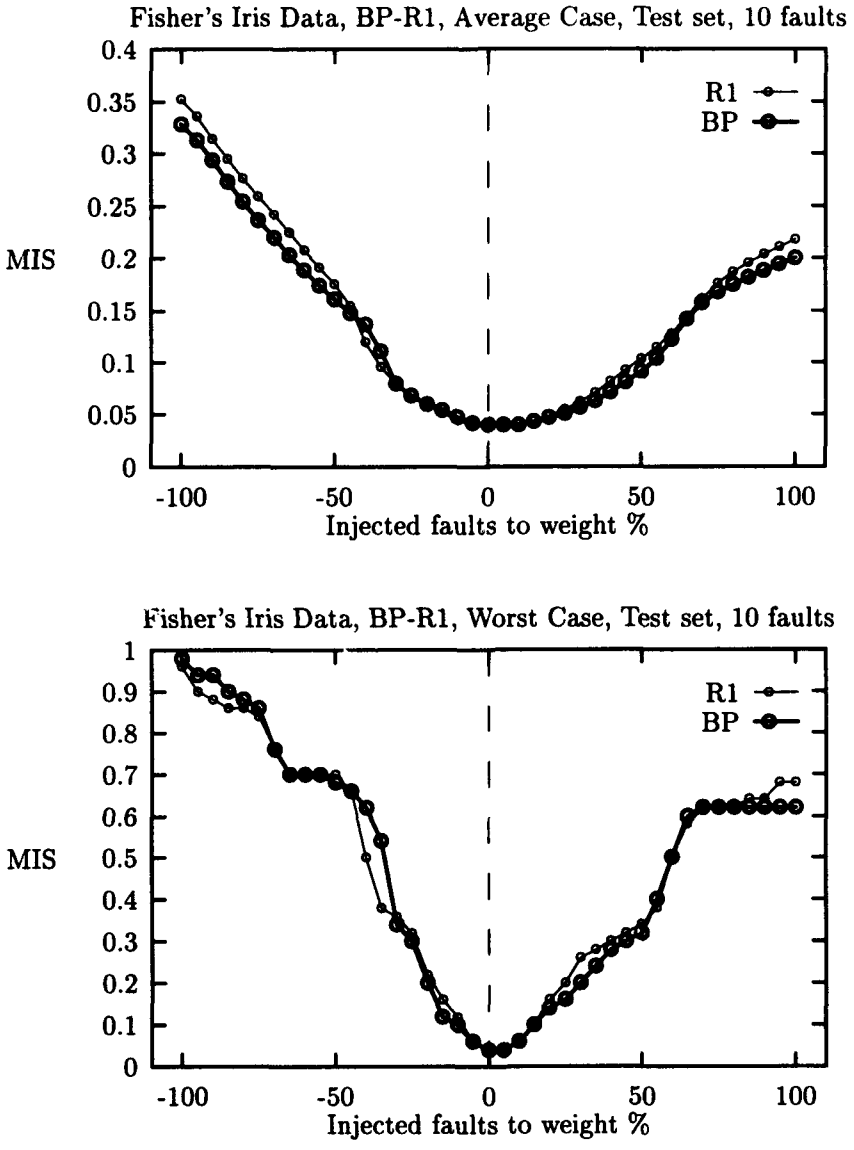


Figure 22: Comparison of $\mathcal{N}_{BP}$ and $\mathcal{N}_{R_1}$ on the MIS of Fisher's data on test set using multiple faults evaluation.

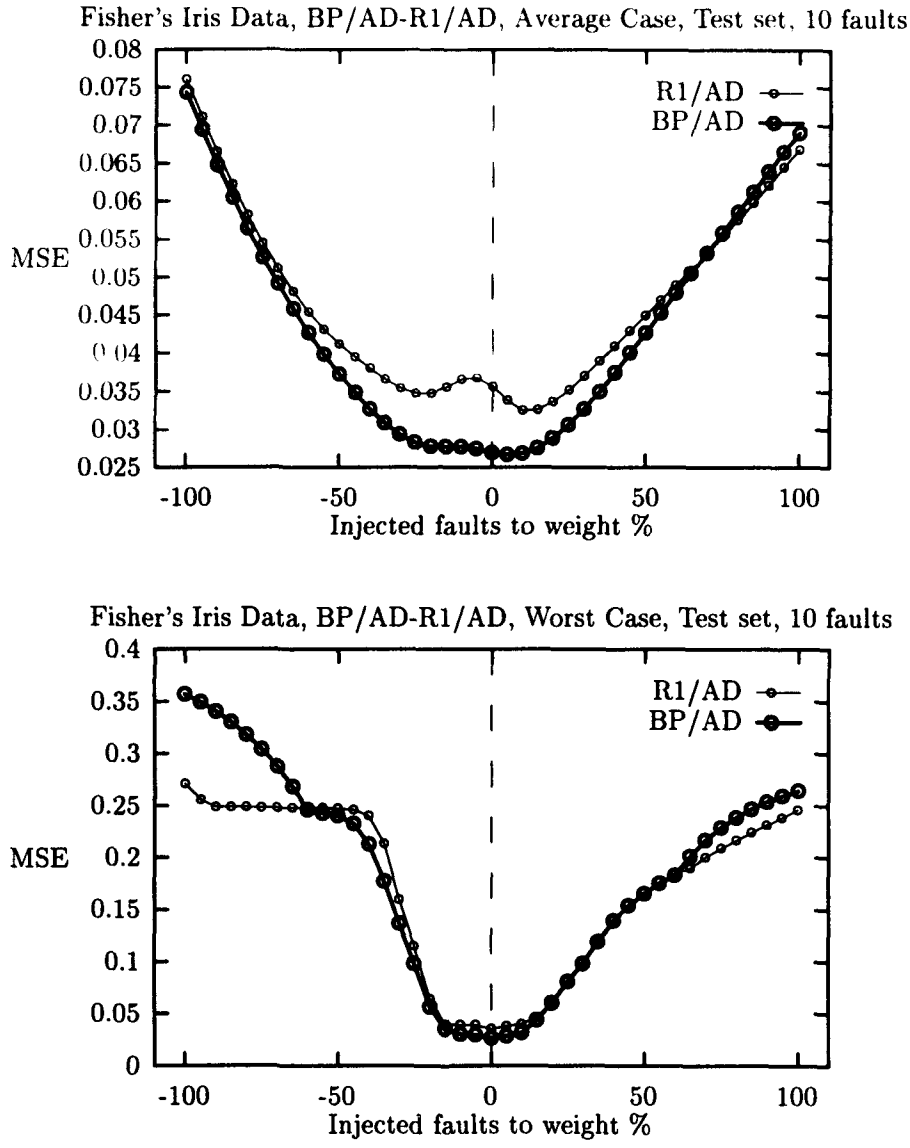


Figure 23: Comparison of $\mathcal{N}_{BP-AD}$ and $\mathcal{N}_{R_1-AD}$ on the MSE of Fisher's data on test set using multiple faults evaluation.

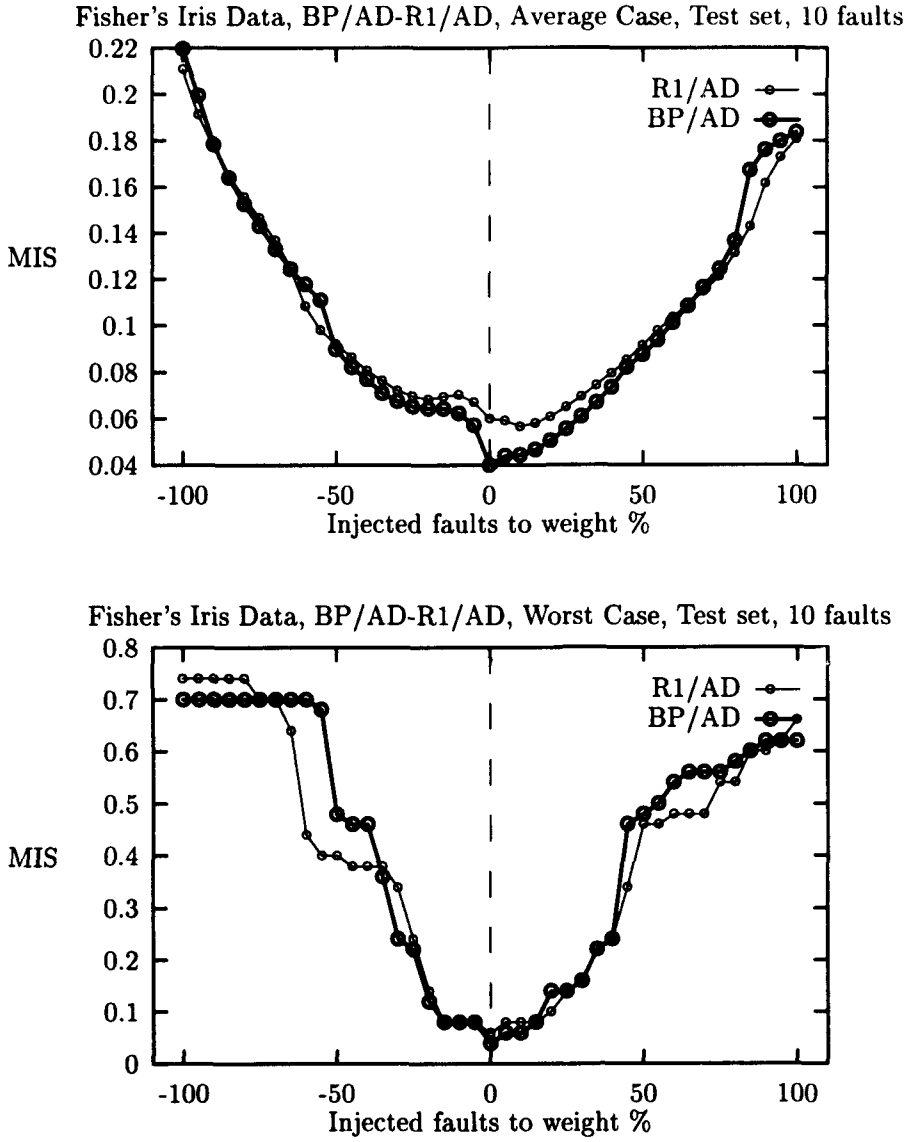


Figure 24: Comparison of $\mathcal{N}_{BP-AD}$ and $\mathcal{N}_{R_1-AD}$ on the MIS of Fisher's data on test set using multiple faults evaluation.

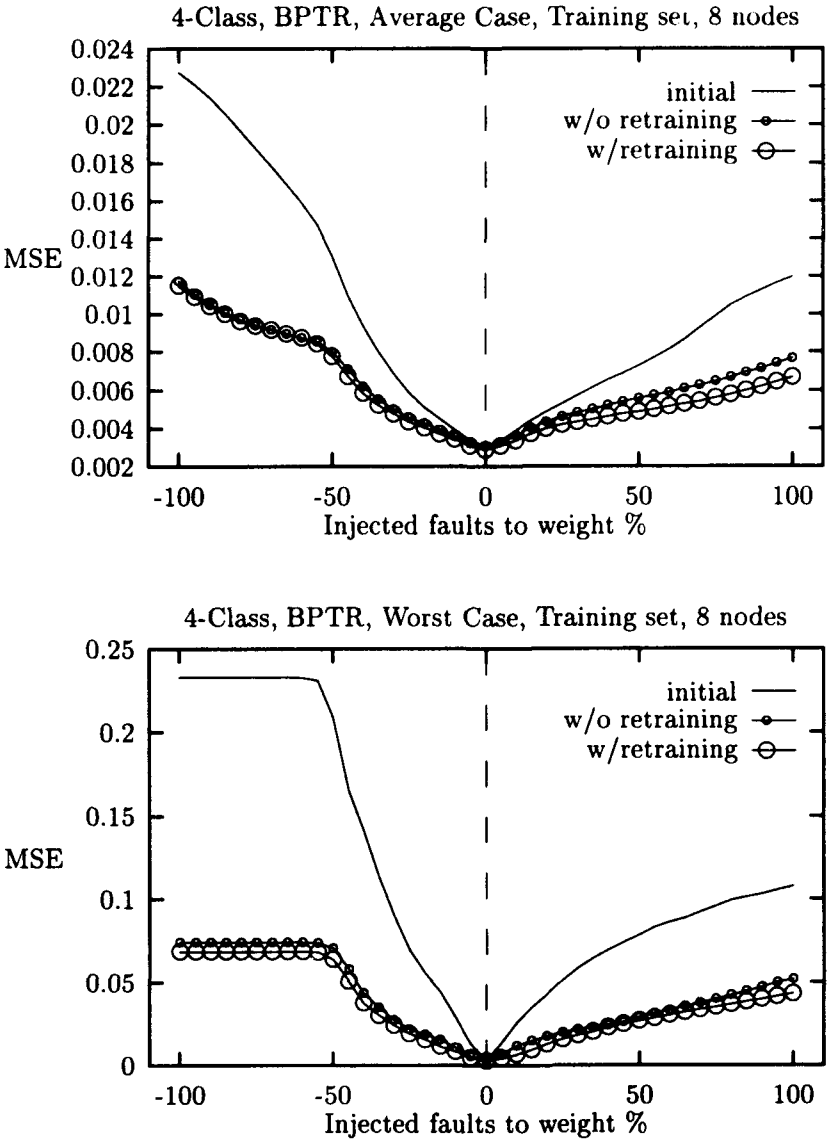


Figure 25: Comparison of *BP* with and without further retraining after adding node on the MSE of 4-class discrimination data on training set.

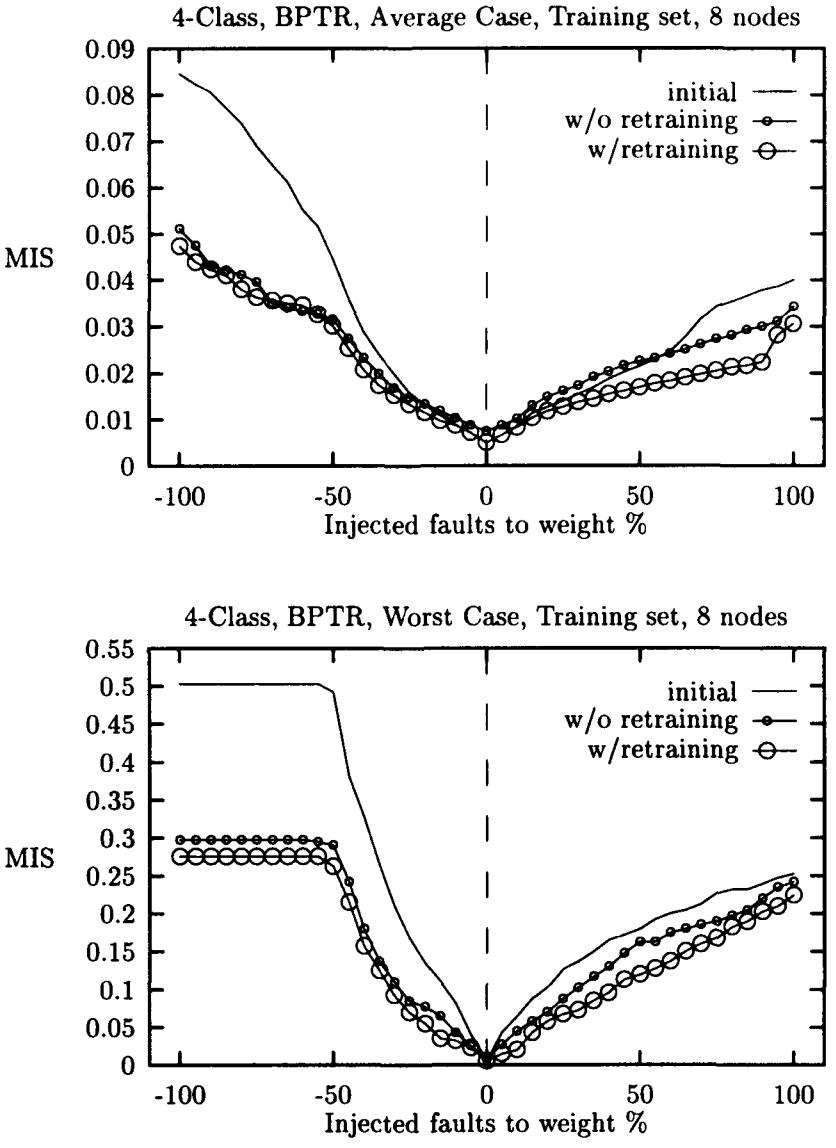


Figure 26: Comparison of BP with and without further retraining after adding node on the MIS of 4-class discrimination data on training set.

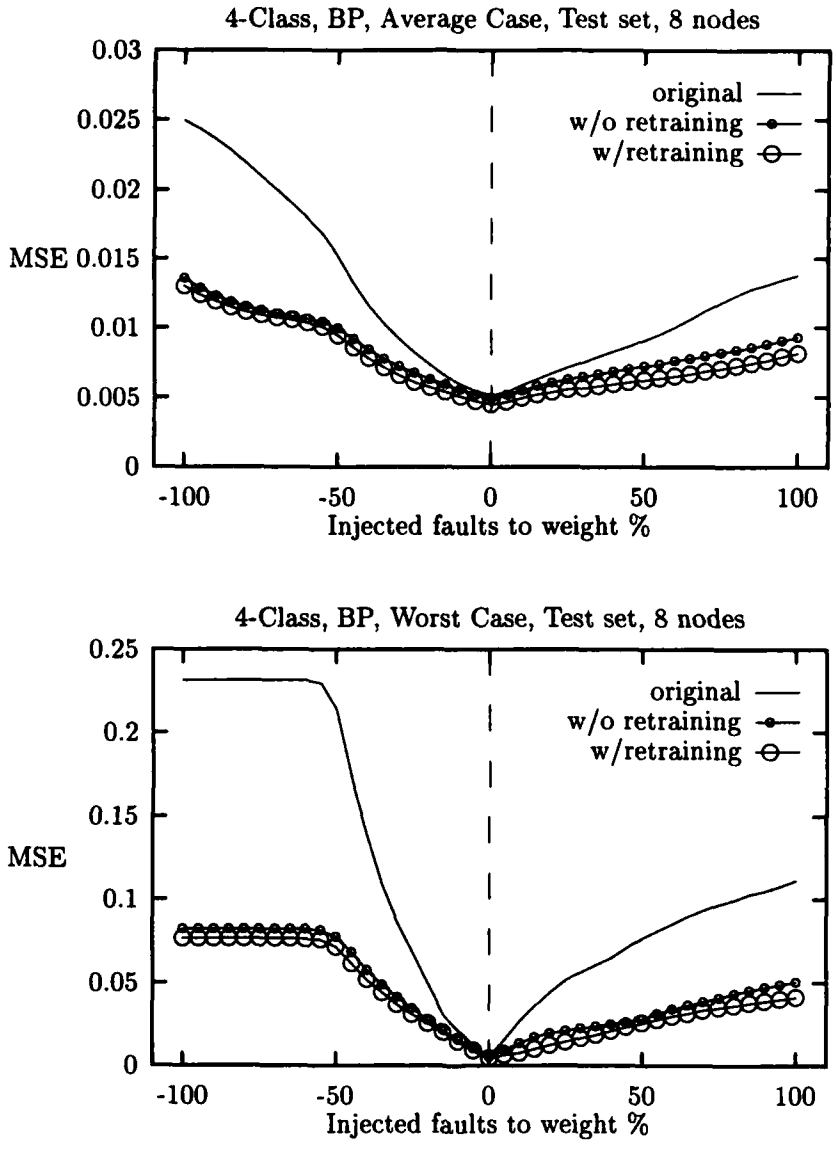


Figure 27: Comparison of *BP* with and without further retraining after adding node on the MSE of 4-class discrimination data on test set.

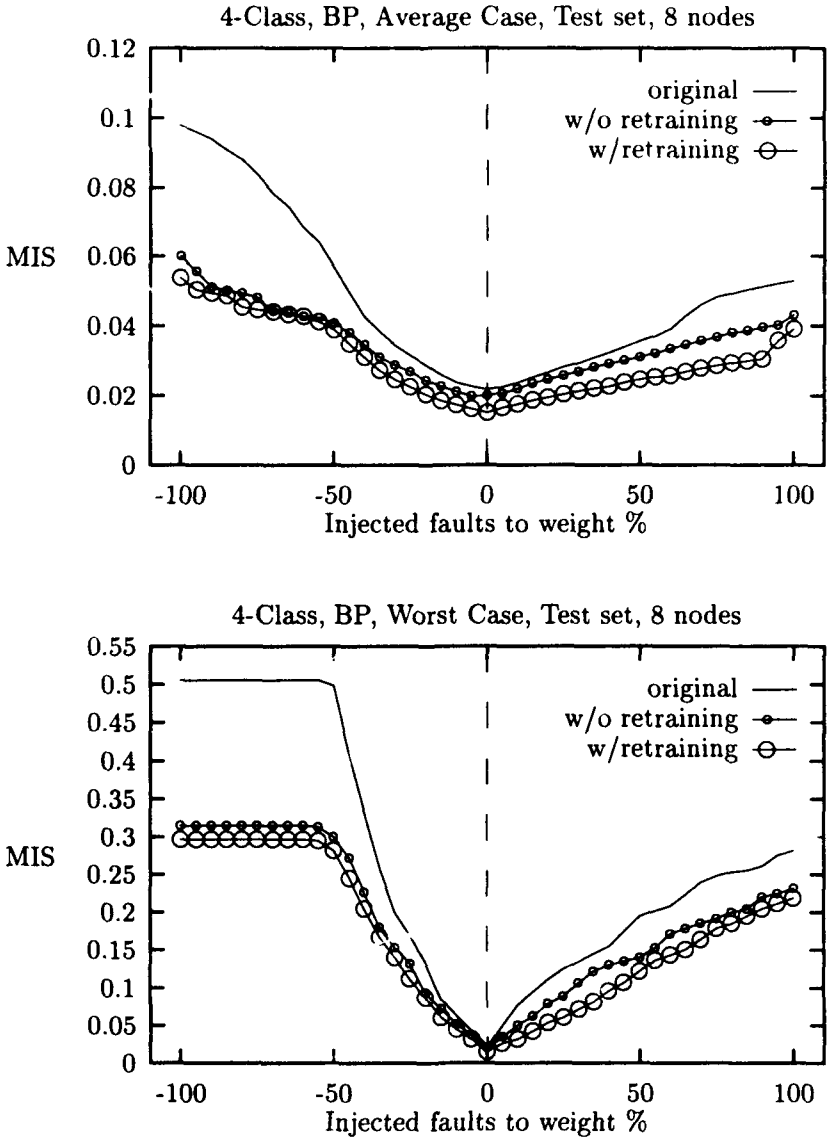


Figure 28: Comparison of BP with and without further retraining after adding node on the MIS of 4-class discrimination data on test set.

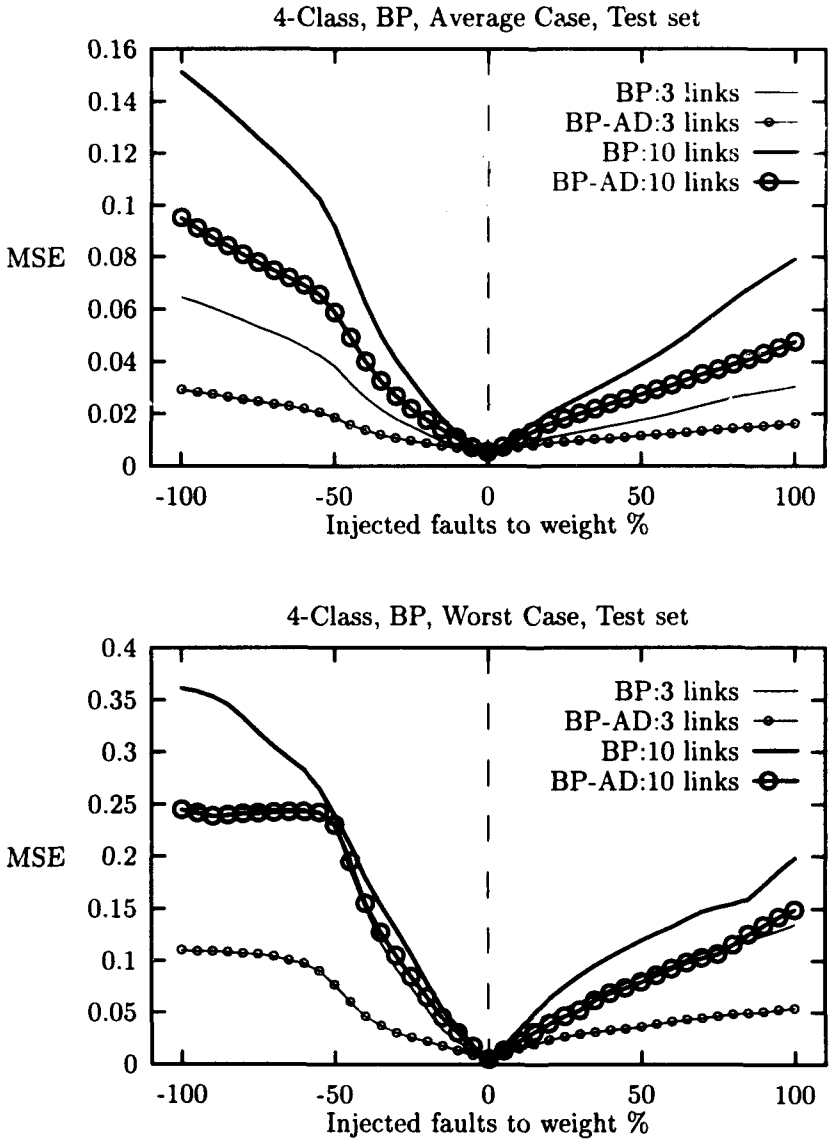


Figure 29: Comparison of $\mathcal{N}_{BP}$ and $\mathcal{N}_{BP-AD}$ on the MSE of 4-class data on test set using multiple faults evaluation.

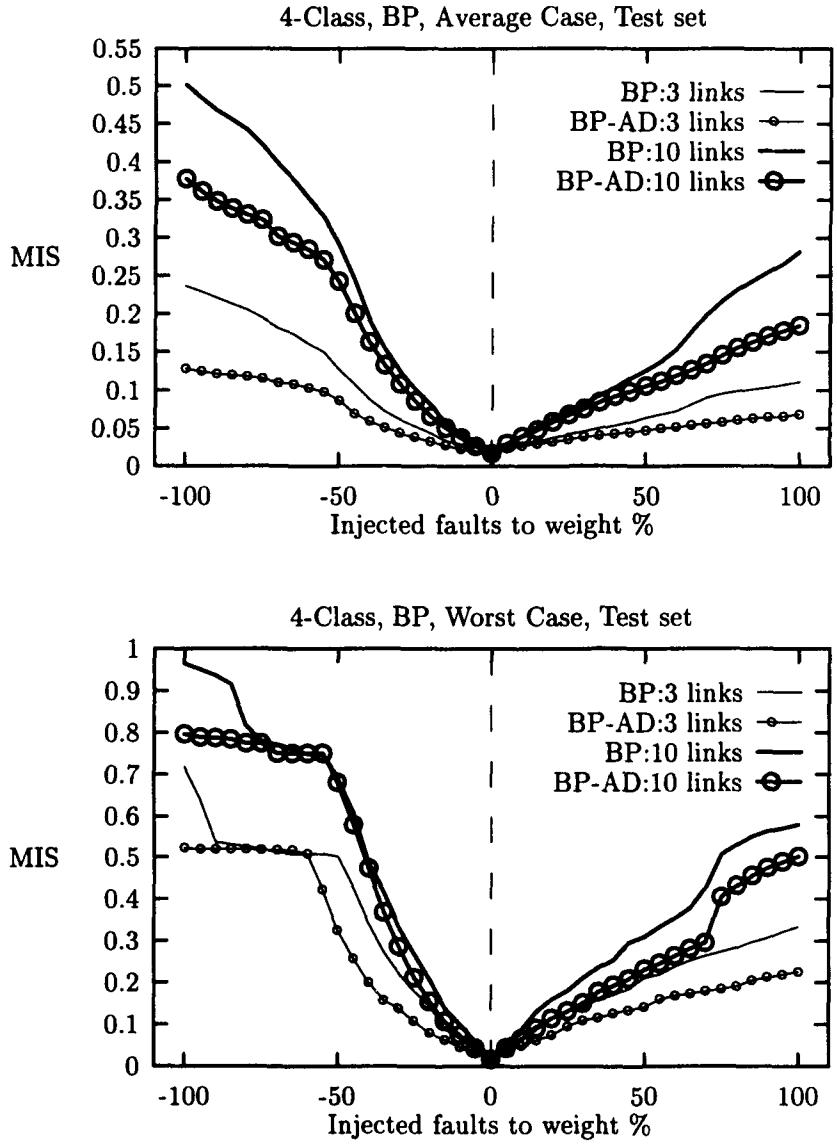


Figure 30: Comparison of $\mathcal{N}_{BP}$ and $\mathcal{N}_{BP-AD}$ on the MIS of 4-class data on test set using multiple faults evaluation.

Performance degradations of the initial and final 4-10-3 networks are shown in Table 2 and Figure 31. Our robustness procedure achieved 83% improvement on average sensitivity and 81% improvement on worst sensitivity for this problem.

On both average case and worst case for the training set, the robustness of BP and R_1 are similar both in initial state and after applying A/D. For the test set, BP and R_1 both obtain similar improvements in robustness. The MSE in this case is increased because in both networks we have further retraining which causes over-fitting on the training set and yields poor generalization on the test set.

Figures 13 to 16 show the comparative results of BP without further retraining after adding a node and with further retraining after adding a node. For the training set, MSE does decrease after further retraining, but the robustness which is gained from A/D is destroyed at the small scales in the average case. In Figure 13, although some robustness is lost, the MSE is decreased by a large amount which makes the loss of robustness irrelevant. For the test set, over-training results in poor generalization, and both robustness and performance are worse than for the one without further retraining. See Figure 15 and Figure 16.

Figures 17 to 24 show the several comparisons using the evaluation method of multiple faults.

8.5.2 Four-Class Grid Discrimination Problem

In this problem we classify a given two-dimensional observation as belonging to one of four classes. The training sample consisted of 400 exemplars, randomly and uniformly generated to fall into 4 classes. The test set consisted of 600 more similarly generated patterns. The starting neural network was of size 2-8-4, it was reduced to 2-4-4 in the first stage of the algorithm, and was built up to 2-8-4. Results are given in Table 3 for the test set only. Our robustness procedure achieved a 71% improvement on average sensitivity and 69% improvement on worst sensitivity for this problem.

On both the average case and the worst case for the training set, the robustness of BP and R_1 are similar both in initial state and after applying A/D. For the test set, BP and R_1 both obtain similar improvements in robustness. The MSE in this case is increased because in both networks we have further retraining, which causes over-fitting on the training set and yields poor generalization on the test set.

Figures 25 to 28 show the comparative results of BP without further retraining after adding a node and with further retraining after adding a node. These figures show that further retraining improves performance.

Figures 29 and 30 show the comparison using the evaluation method of multiple faults. We have compared N_{BP} *vs.* $N_{BP\text{-}AD}$ on 3-link failures and 10-link failures, respectively.

	Initial Net	Final Net
Hidden nodes	10	10
Training MSE	0.005130	0.005129
Testing MSE	0.025139	0.022123
Training Correctness(%)	99.00	99.00
Testing Correctness(%)	94.00	96.00
Avg. Sensitivity	0.025686	0.004314
Wst. Sensitivity	0.110977	0.021464

Table 2: Results of Fisher's Iris data on Combination 0. Deletion/addition process is $10 \rightarrow 4 \rightarrow 5 \rightarrow 6 \rightarrow 7 \rightarrow 8 \rightarrow 9 \rightarrow 7 \rightarrow 8 \rightarrow 9 \rightarrow 10$.

	Initial Net	Final Net
Hidden nodes	8	8
Training MSE	0.003038	0.003038
Testing MSE	0.005132	0.004797
Training Correctness(%)	99.50	99.25
Testing Correctness(%)	97.83	97.83
Avg. Sensitivity	0.091697	0.026260
Wst. Sensitivity	0.229797	0.071189

Table 3: Results of 4-class discrimination problem on Combination 0. The deletion/addition process is $8 \rightarrow 4 \rightarrow 5 \rightarrow 6 \rightarrow 7 \rightarrow 5 \rightarrow 6 \rightarrow 7 \rightarrow 8$.

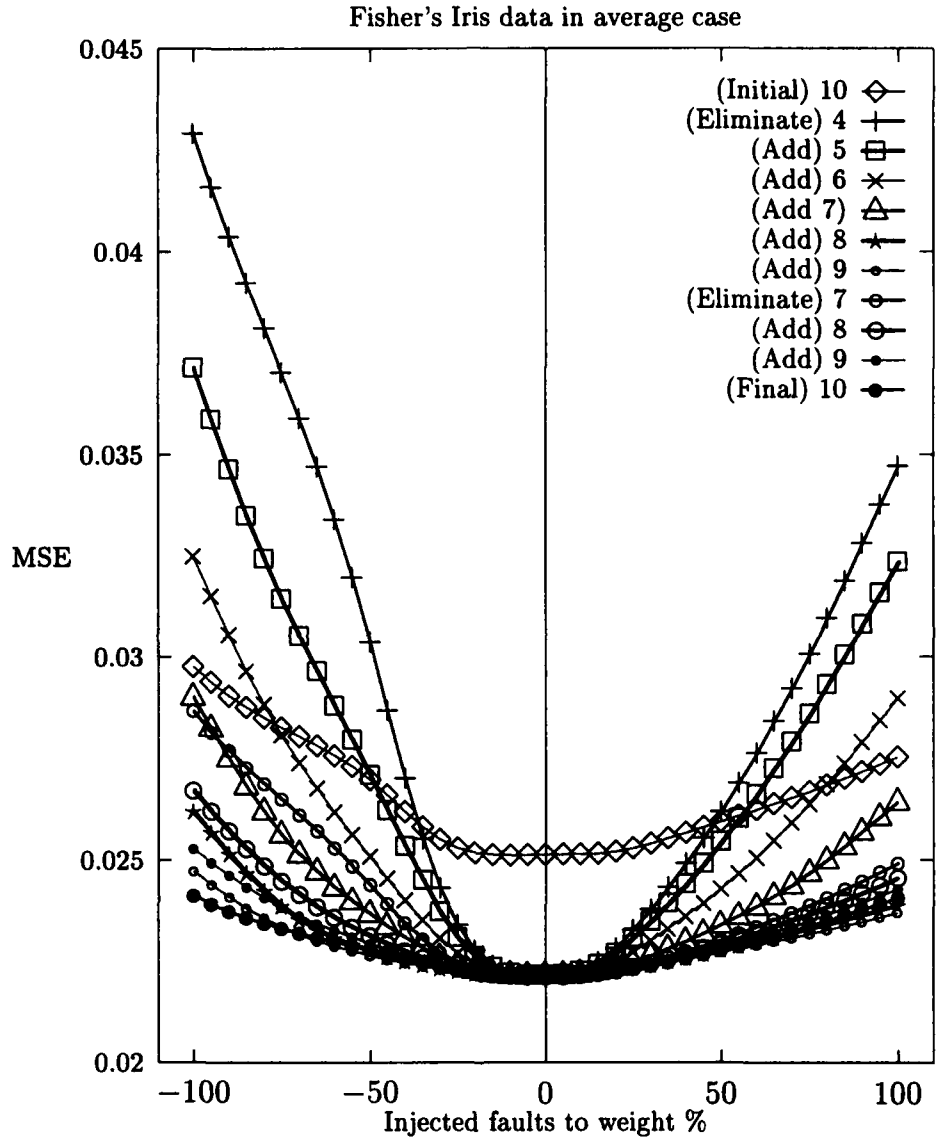


Figure 31: The above graph shows all curves of deletion/addition process using combination 0. The sequence shown is $10 \rightarrow 4 \rightarrow 5 \rightarrow 6 \rightarrow 7 \rightarrow 8 \rightarrow 9 \rightarrow 7 \rightarrow 8 \rightarrow 9 \rightarrow 10$, indicating the number of hidden nodes. Observe that the original 10-node network is roughly as robust as a 6-node network for high perturbations and worse than all other cases for small perturbations.

8.5.3 Two-Character Recognition Data

The two letters "A" and "B" were selected from Letter Image Recognition Data created by David J. Slate. The original 26 letters data was used by Slate to investigate the ability of several variations of Holland-style adaptive classifier systems to learn to correctly guess the letter categories associated with vectors of 16 integer attributes extracted from raster scan images of the letters. The best accuracy obtained was a little over 80%. The objective is to identify each of a large number of black-and-white rectangular pixel displays as one of the 26 capital letters in the English alphabet. The character images were based on 20 different fonts and each letter within these 20 fonts was randomly distorted to produce a file of 20,000 unique stimuli. Each stimulus was converted into 16 primitive numerical attributes (statistical moments and edge counts) which were then scaled to fit into a range of integer values from 0 through 15.

We select 500 instances out of 1555 instances as a training set, and leave the others as a test set. There are 262 instances for "A" and 238 instances for "B" in the training set. The configuration of neural network is 16-15-1 for the input, hidden and output layers, respectively.

For convenience, we show the node sensitivities of N_{BP} and N_{RI} in Table 4. This table shows that most of the nodes (13 out of 15) N_{RI} are insensitive and will be removed when A/D is applied. On the other hand, N_{BP} only removed 4 nodes out of 15 nodes. So we can see that the robustness improvement in N_{RI} is much better than N_{BP} because N_{RI} added more than three times the number of nodes added by N_{BP} to the original network.

In the absence of further retraining, we continue to add nodes until the improvement in the network's robustness is negligible, or until the number of nodes reached equals the number of nodes in the initial network, to allow fair comparison of the robustness of networks of equal size. Other possible criteria can also be used, such as adding extra nodes until the sensitivity of the current most critical node is less than some proportion of the sensitivity of the initial most critical node. The termination criterion for retraining, if conducted after adding nodes, is to cease retraining when improvement in MSE is negligible.

8.6 Results for Other Measures of Sensitivity

This subsection shows the experimental results of combination 1, 2 and 3, on Fisher's data and the 4-class problem.

8.6.1 Results for Combination 1

Results for Fisher's data are shown in Table 5 and Figures 32 and 33. Results for the 4-class problem are shown in Table 6 and Figures 34 and 35.

Node j	$\mathcal{N}_{BP}$	$\mathcal{N}_{R_1}$
0	0.011100	0.000516
1	0.308812	0.000516
2	0.033333	0.000516
3	0.294912	0.000516
4	0.479885	0.343697
5	0.318664	0.000516
6	0.283962	0.033032
7	0.264837	0.000516
8	0.487141	0.000516
9	0.480943	0.313870
10	0.000008	0.000516
11	0.361462	0.000516
12	0.000043	0.000516
13	0.299816	0.000516
14	0.369992	0.000516
MSE	0.018012	0.098332

Table 4: Node sensitivities of $\mathcal{N}_{BP}$ and $\mathcal{N}_{R_1}$ after training and before applying A/D for two-character recognition data with 15 hidden nodes.

	Initial Net	Final Net
Hidden nodes	10	10
Training MSE	0.005130	0.005129
Testing MSE	0.025139	0.022988
Training Correctness(%)	99.00	99.00
Testing Correctness(%)	94.00	96.00
Avg. Sensitivity	0.004074	0.002652
Wst. Sensitivity	0.010754	0.009945

Table 5: Results of Fisher's data problem on Combination 1. The deletion/addition process is $10 \rightarrow 6 \rightarrow 7 \rightarrow 8 \rightarrow 6 \rightarrow 7 \rightarrow 8 \rightarrow 9 \rightarrow 10$.

	Initial Net	Final Net
Hidden nodes	8	8
Training MSE	0.003038	0.003038
Testing MSE	0.005132	0.005067
Training Correctness(%)	99.50	99.00
Testing Correctness(%)	97.83	98.00
Avg. Sensitivity	0.004266	0.001663
Wst. Sensitivity	0.008332	0.005135

Table 6: Results of 4-class discrimination problem on Combination 1. The deletion/addition process is $8 \rightarrow 7 \rightarrow 8 \rightarrow 7 \rightarrow 8$.

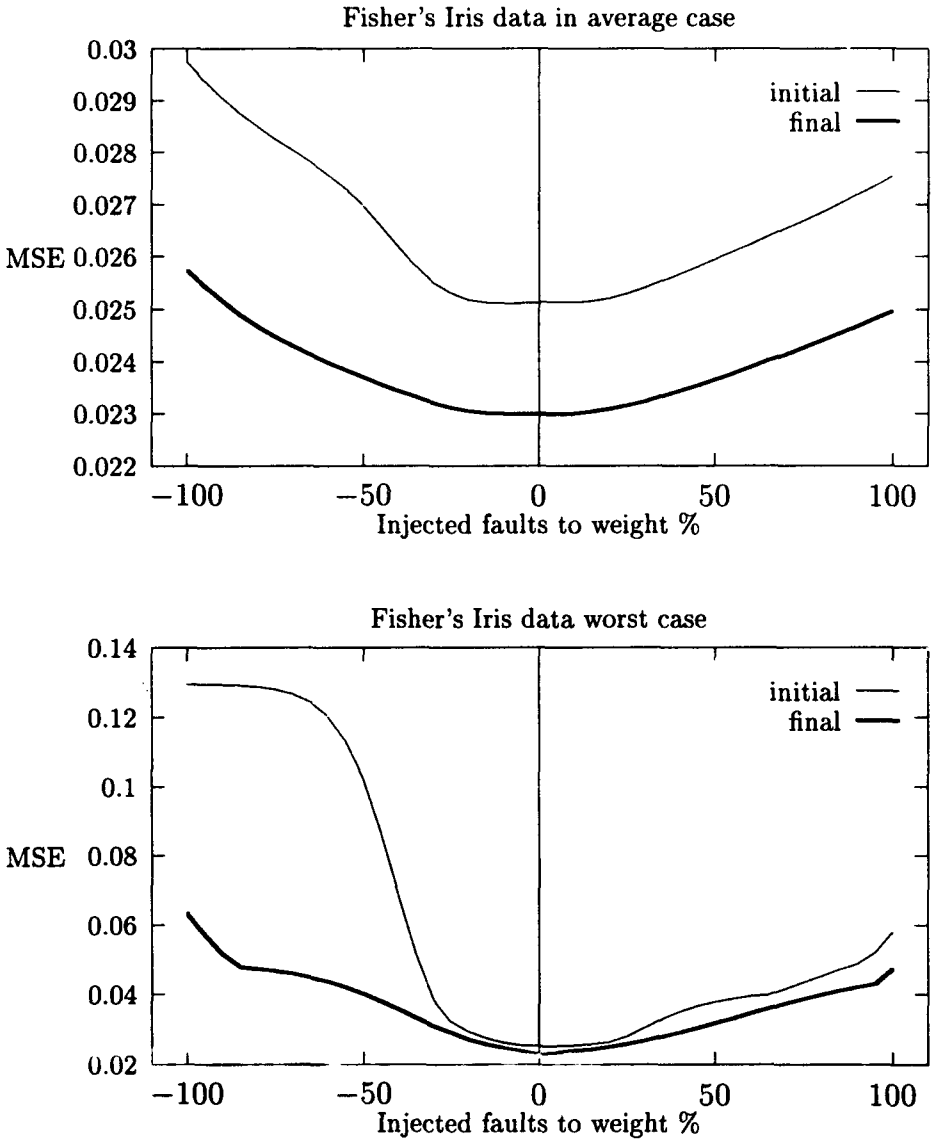


Figure 32: Results of Fisher's Iris data on Combination 1 with 10 hidden nodes for test set measured in MSE.

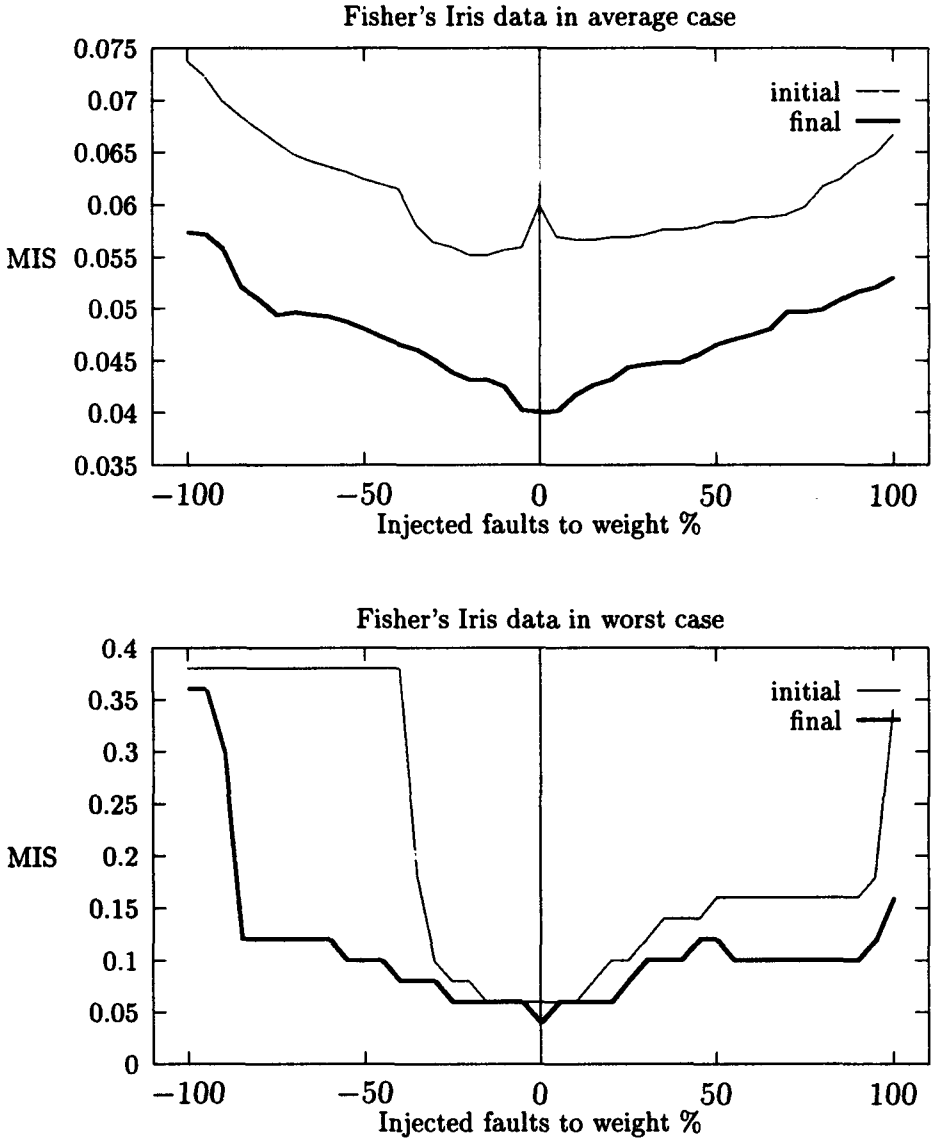


Figure 33: Results of Fisher's Iris data on Combination 1 with 10 hidden nodes for test set measured in MIS.

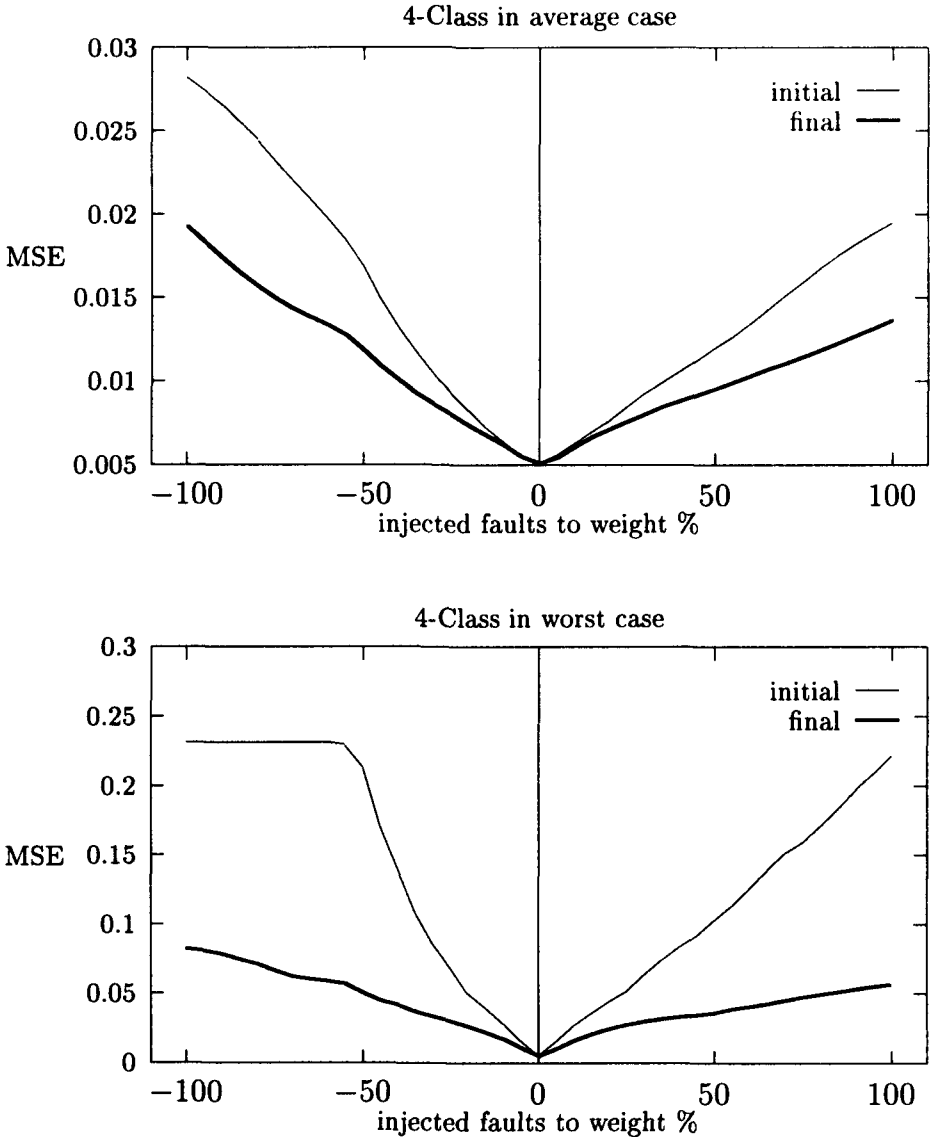


Figure 34: Results of 4-class discrimination problem on Combination 1 with 8 hidden nodes, for the test set, measured in MSE.

8.6.2 Results for Combination 2

Results for Fisher's data are shown in Table 7 and Figure 36 and 37. Results for 4-class problem are shown in Table 8 and Figure 38 and 39.

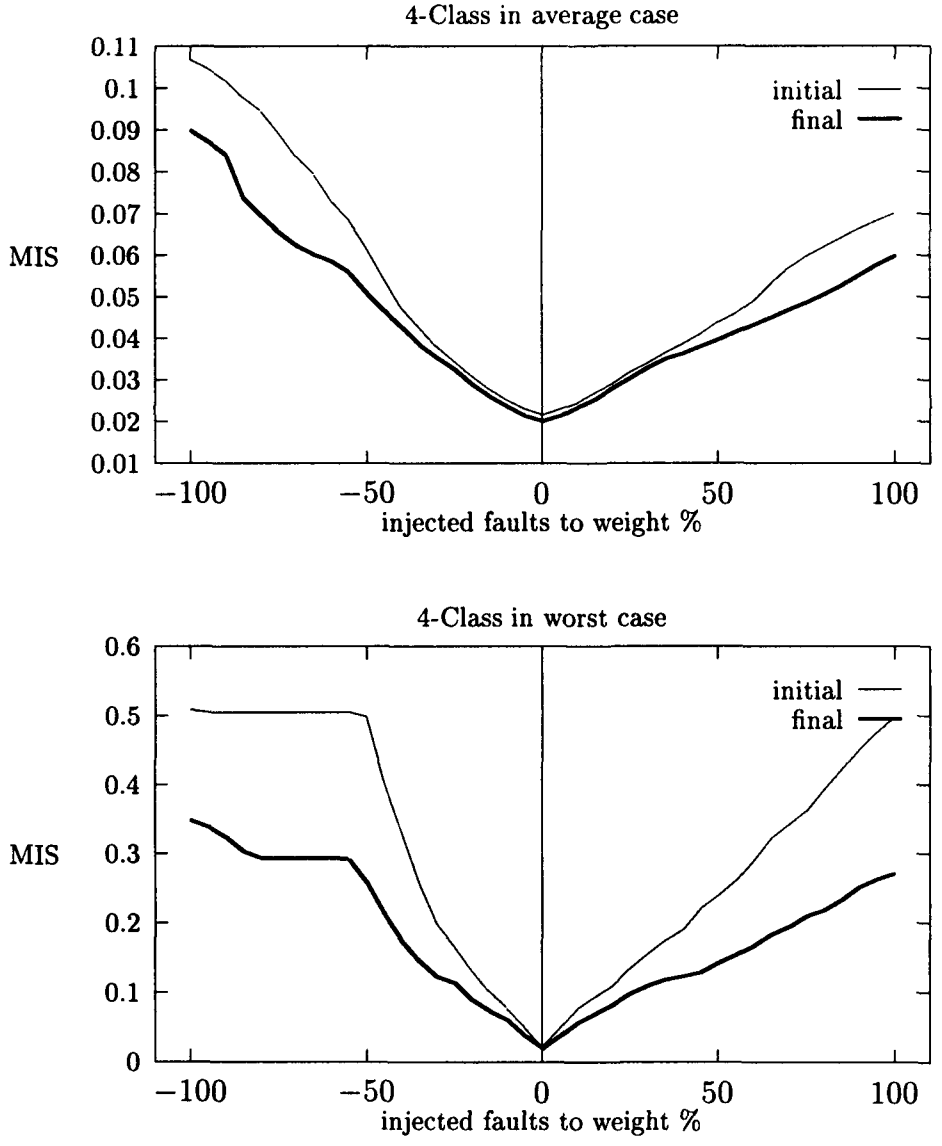


Figure 35: Results of 4-class discrimination problem on Combination 1 with 8 hidden nodes, for the test set, measured in MIS.

	Initial Net	Final Net
Hidden nodes	10	5
Training MSE	0.005130	0.111194
Testing MSE	0.025139	0.143500
Training Correctness(%)	99.00	66.00
Testing Correctness(%)	94.00	62.00
Avg. Sensitivity	0.000296	0.001382
Wst. Sensitivity	0.001414	0.005251

Table 7: Results of Fisher's data problem on Combination 2. The deletion/addition process is $10 \rightarrow 4 \rightarrow 5 \rightarrow 4 \rightarrow 5 \rightarrow 4 \rightarrow 5$.

	Initial Net	Final Net
Hidden nodes	8	8
Training MSE	0.003038	0.003038
Testing MSE	0.005132	0.004799
Training Correctness(%)	99.50	99.25
Testing Correctness(%)	97.83	98.00
Avg. Sensitivity	0.001240	0.000220
Wst. Sensitivity	0.005238	0.000843

Table 8: Results of 4-class discrimination problem on Combination 2. The deletion/addition process is $8 \rightarrow 3 \rightarrow 4 \rightarrow 5 \rightarrow 6 \rightarrow 7 \rightarrow 8$.

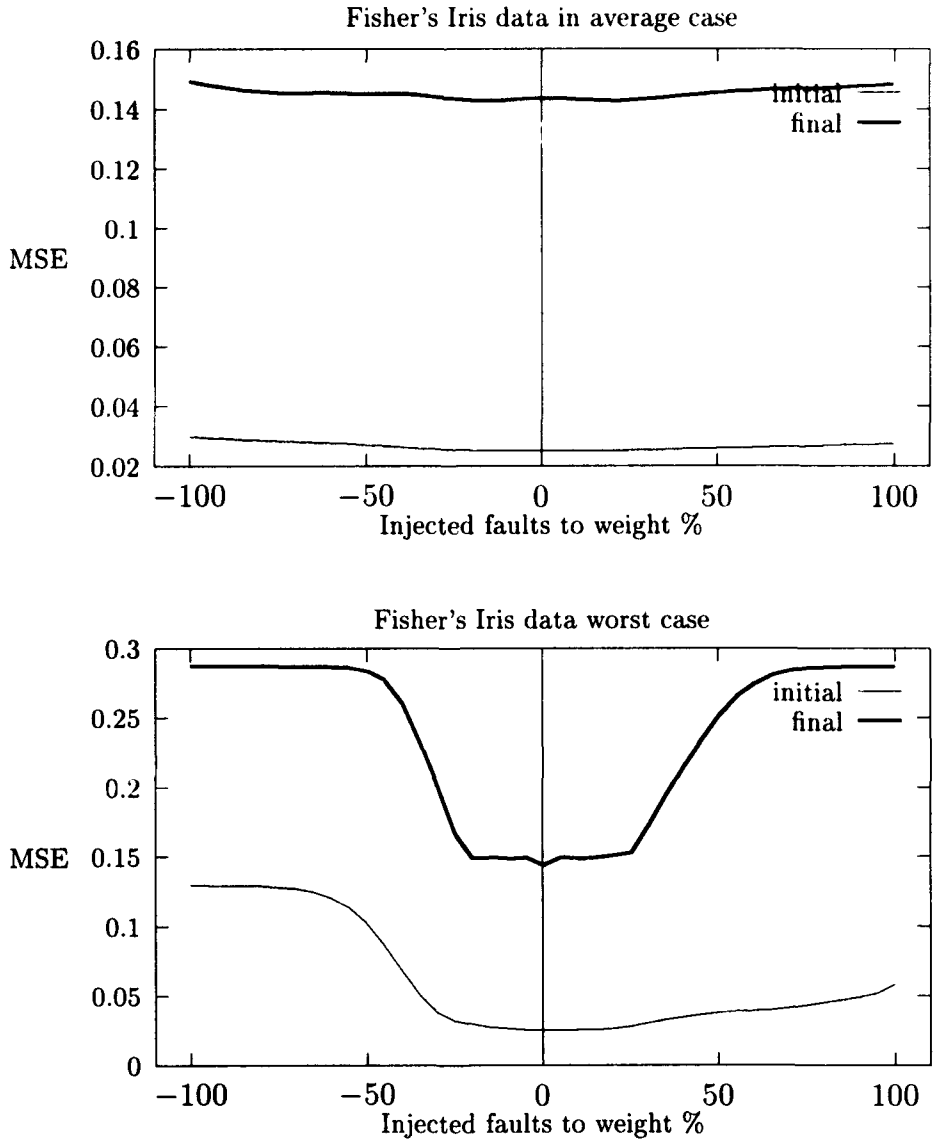


Figure 36: Results of Fisher's Iris data on Combination 2 for test se measured in MSE, with 10 hidden nodes initially and 5 nodes finally.

8.6.3 Results for Combination 3

Results for Fisher's data are shown in Table 9 and Figure 40 and 41. Results for 4-class problem are shown in Table 10 and Figure 42 and 43.

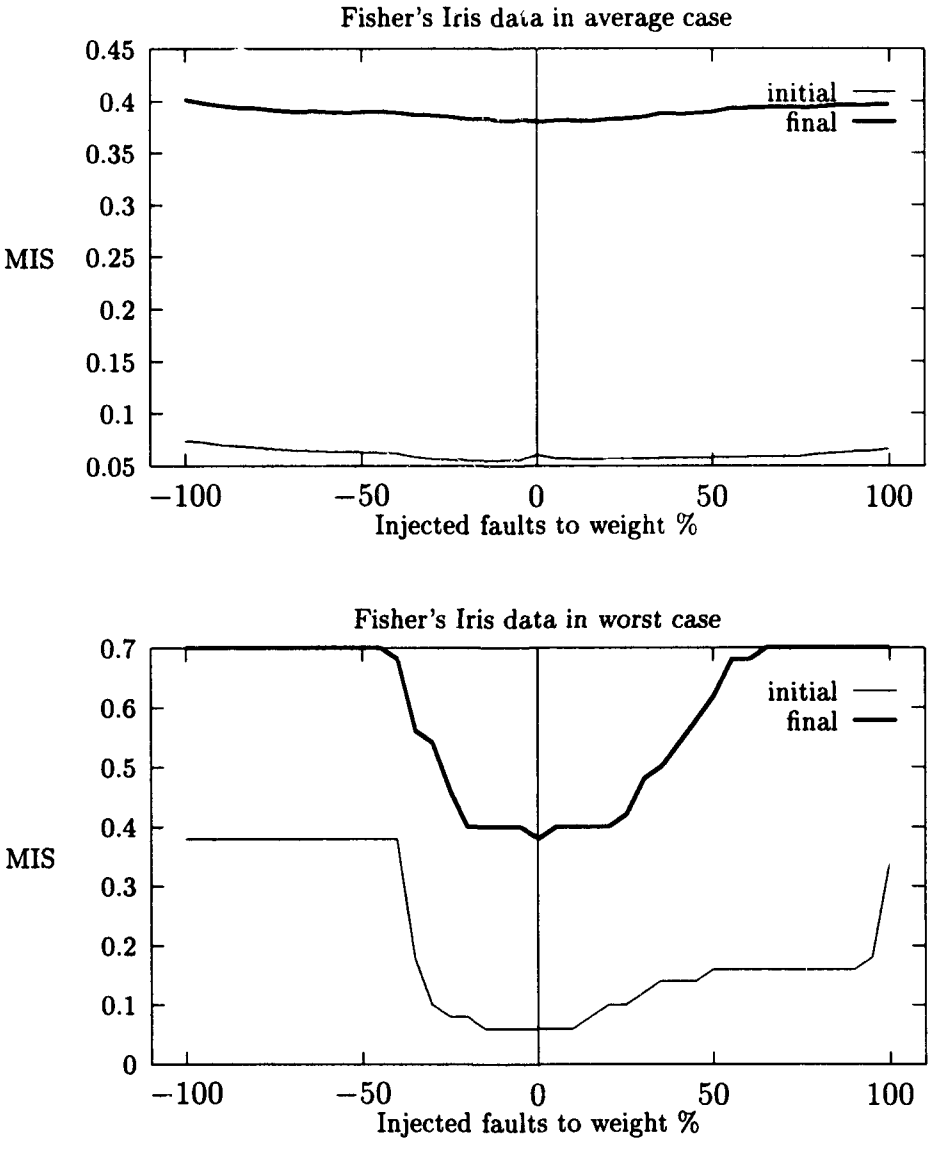


Figure 37: Results of Fisher's Iris data on Combination 2 for test set measured in MIS, with 10 hidden nodes initially and 5 nodes finally.

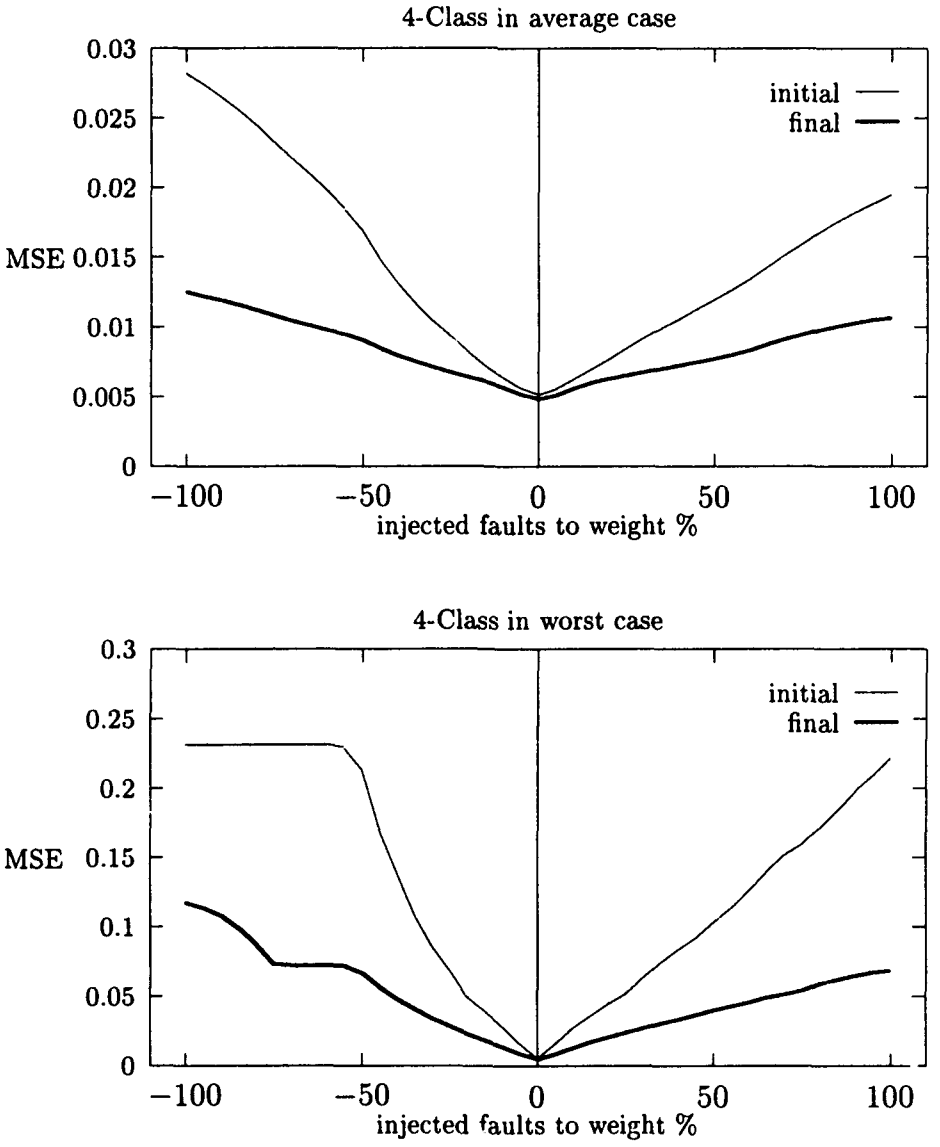


Figure 38: Results of 4-class discrimination problem on Combination 2 with 8 hidden nodes, for the test set, measured in MSE.

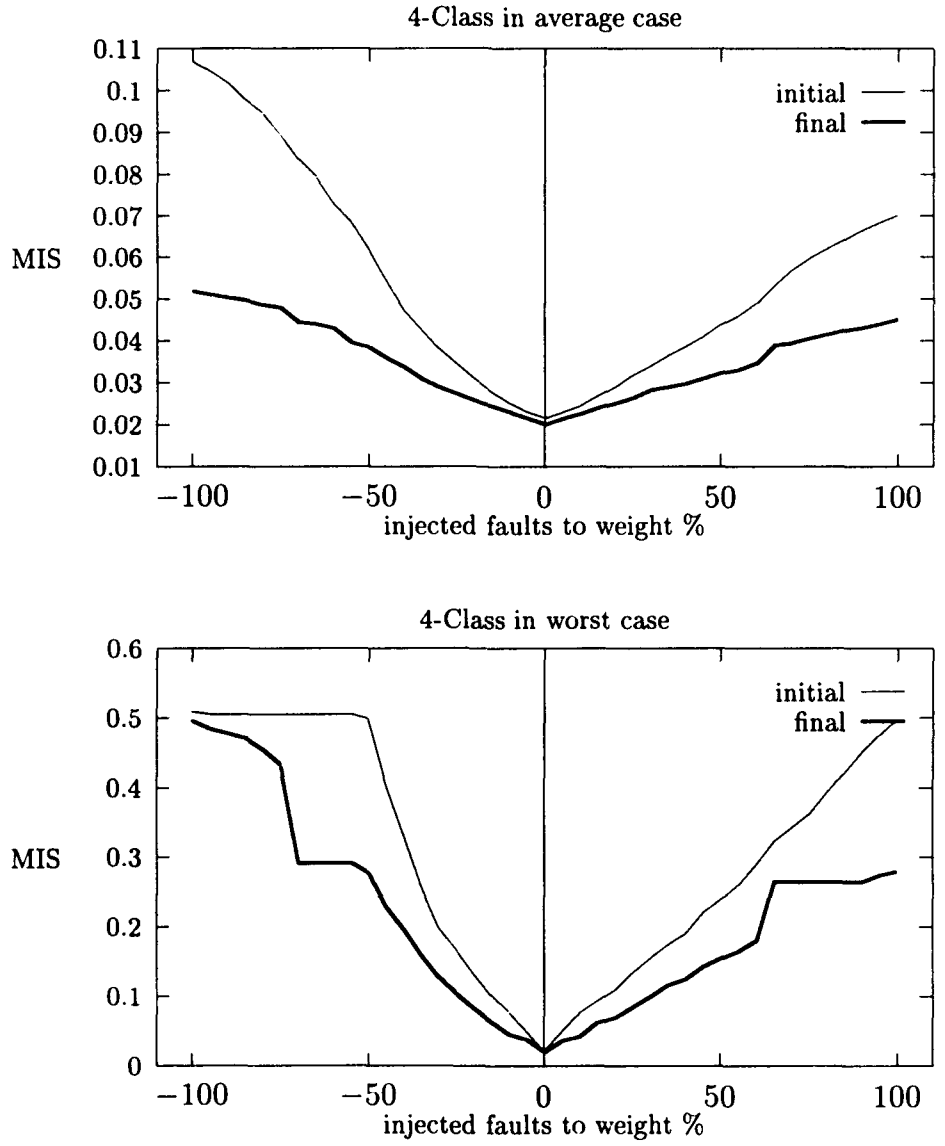


Figure 39: Results of 4-class discrimination problem on Combination 2 with 8 hidden nodes, for the test set, measured in MIS.

	Initial Net	Final Net
Hidden nodes	10	10
Training MSE	0.005130	0.005129
Testing MSE	0.025139	0.024016
Training Correctness(%)	99	9.00
Testing Correctness(%)	94.00	96.00
Avg. Sensitivity	0.001965	0.000936
Wst. Sensitivity	0.005447	0.002975

Table 9: Results of Fisher's data problem on Combination 3. The deletion/addition process is $10 \rightarrow 6 \rightarrow 7 \rightarrow 8 \rightarrow 9 \rightarrow 10$.

	Initial Net	Final Net
Hidden nodes	8	8
Training MSE	0.003038	0.003038
Testing MSE	0.005132	0.005643
Training Correctness(%)	99.50	99.25
Testing Correctness(%)	97.83	97.83
Avg. Sensitivity	0.003173	0.002530
Wst. Sensitivity	0.008718	0.007428

Table 10: Results of 4-class discrimination problem on Combination 3. The deletion/addition process is $8 \rightarrow 7 \rightarrow 8$.

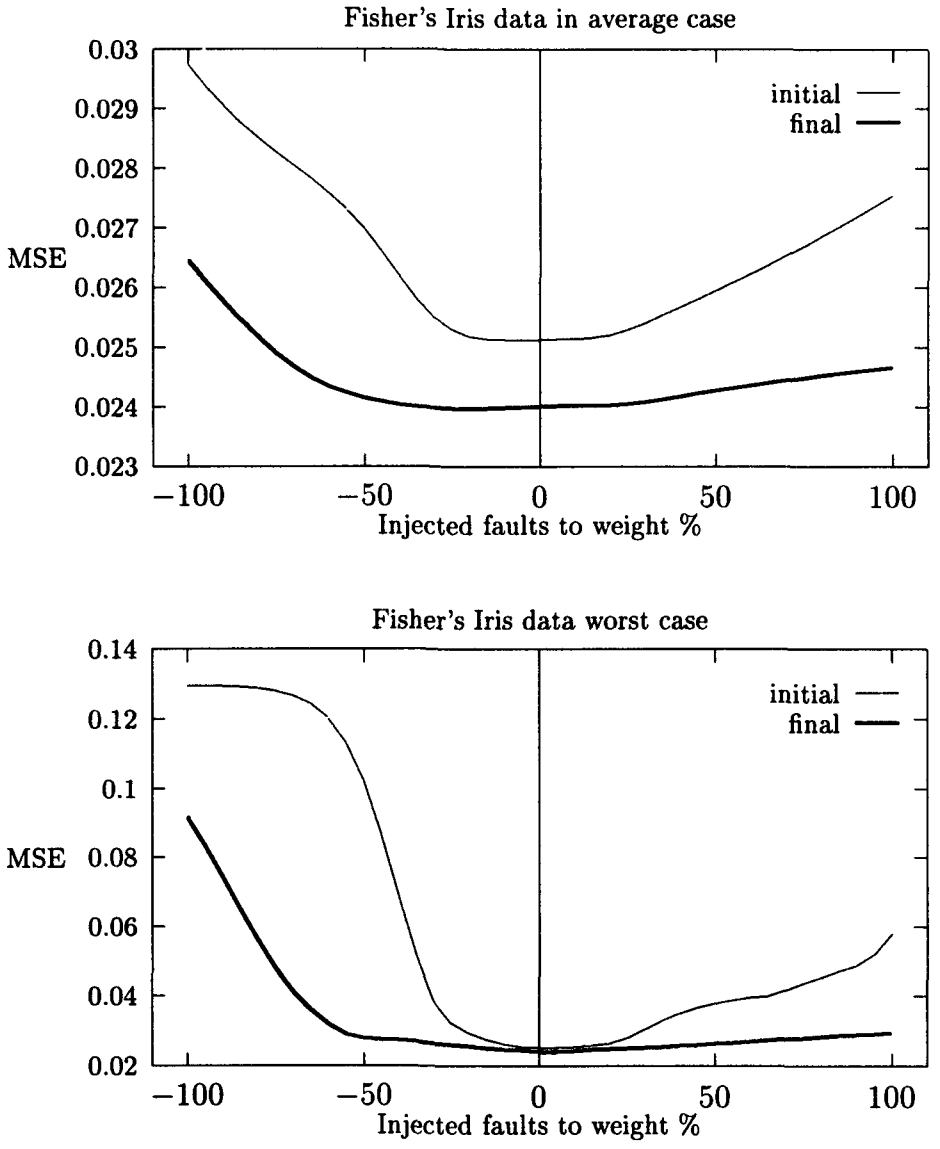


Figure 40: Results of Fisher's Iris data on Combination 3 with 10 hidden nodes for test set measured in MSE.

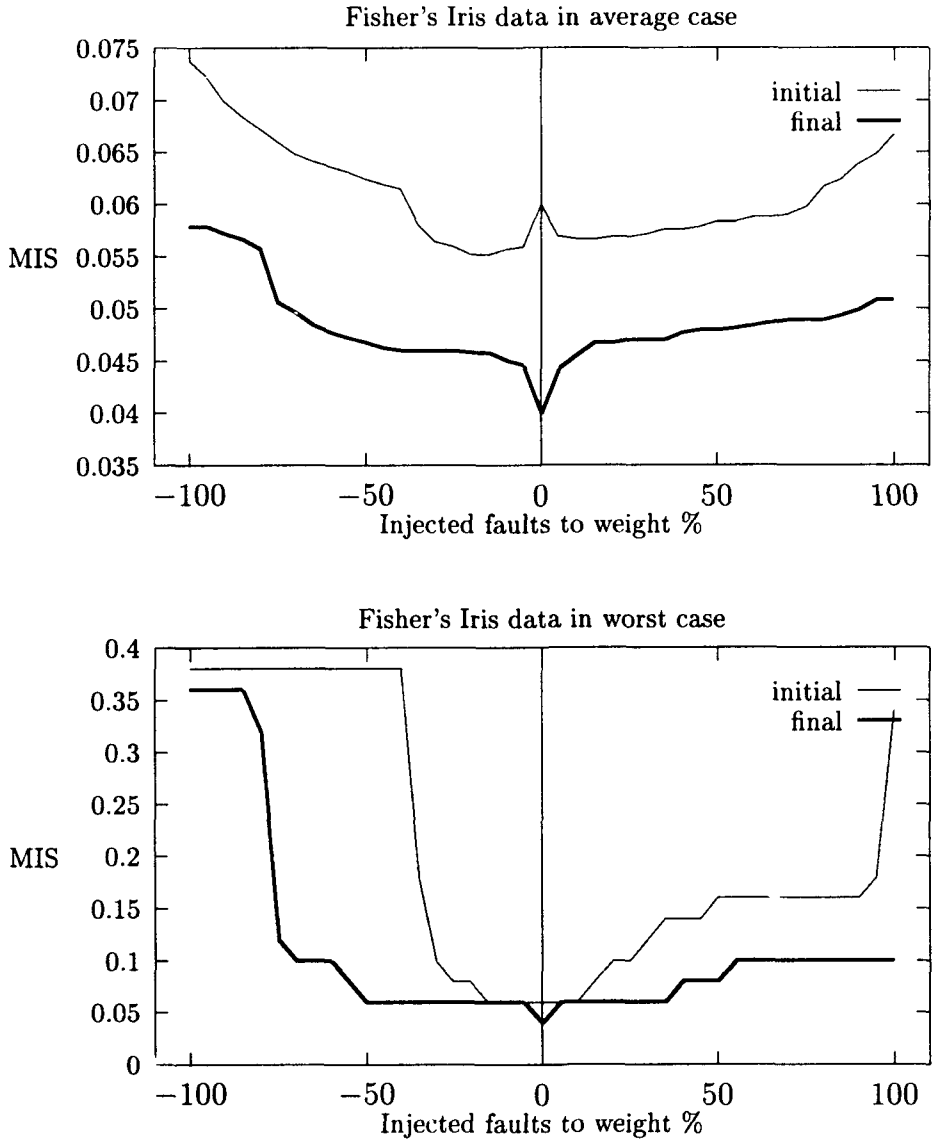


Figure 41: Results of Fisher's Iris data on Combination 3 with 10 hidden nodes for test set measured in MIS.

8.7 Remarks

We have compared the robustness of feedforward neural network training using two different learning rules, BP and R_1. $\mathcal{N}_{R_1}$ has better generalization than $\mathcal{N}_{BP}$ after training, but does not improve fault tolerance when compared to $\mathcal{N}_{BP}$. After applying our new algorithm (A/D), $\mathcal{N}_{R_1}$ will be usually more robust than $\mathcal{N}_{BP}$ because there are more adaptive operations on $\mathcal{N}_{R_1}$. We have investigated the robustness of networks with further retraining after adding nodes in the A/D process. Further retraining will improve the performance on the training

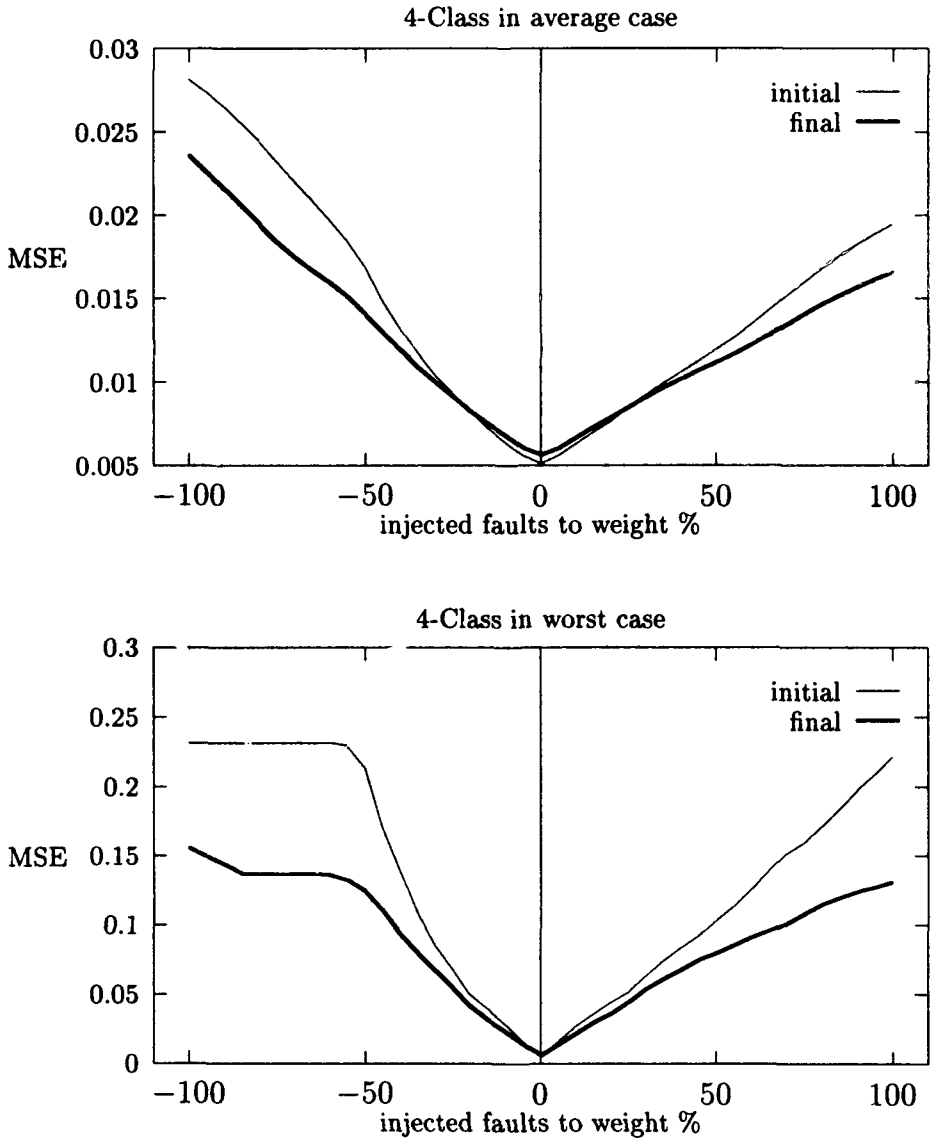


Figure 42: Results of 4-class discrimination problem on Combination 3 with 8 hidden nodes, for the test set, measured in MSE.

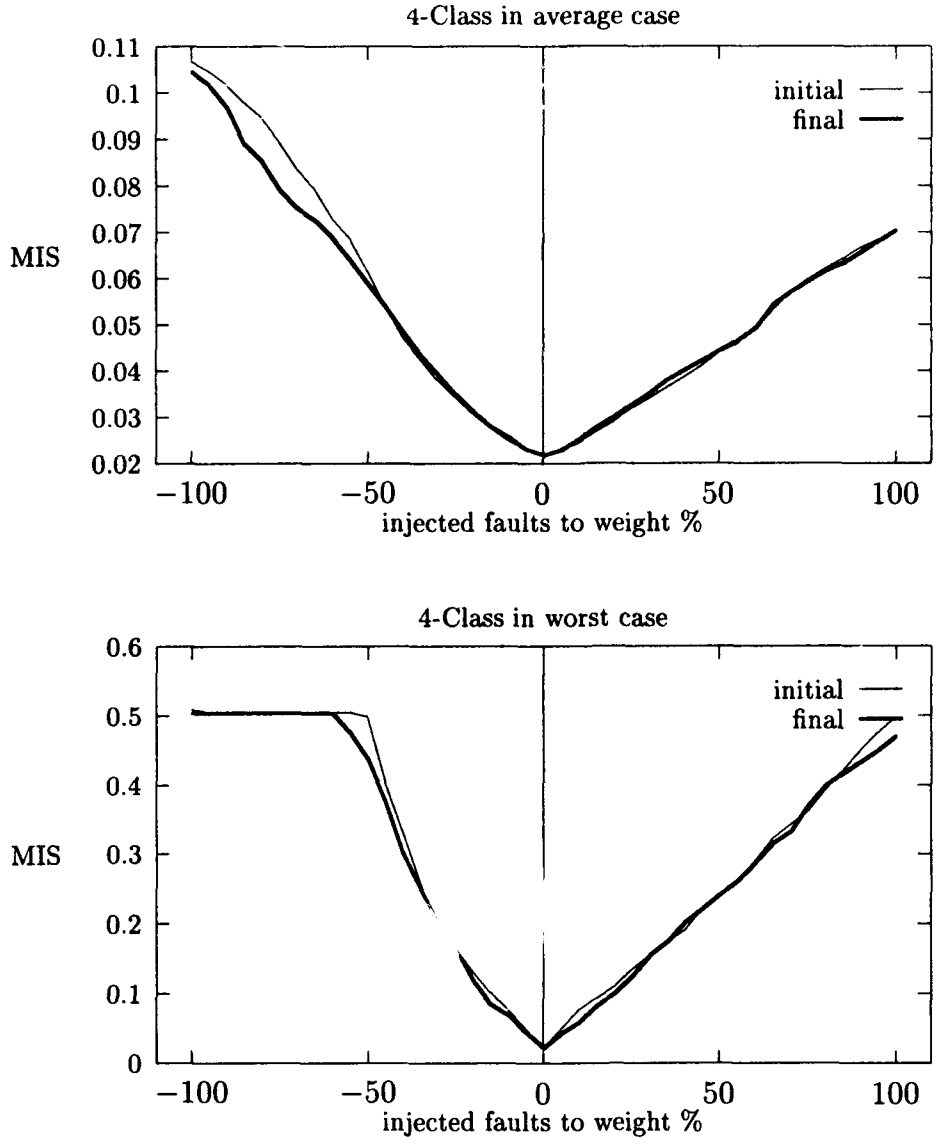


Figure 43: Results of 4-class discrimination problem on Combination 3 with 8 hidden nodes, for the test set, measured in MIS.

set, but destroy the implanted robustness of the network and spoil the generalization when the initial network is well-trained.

We have developed multi-fault sensitivity measures and an algorithm for evaluating multiple faults. This can help us estimate the percentage of faults which the network can tolerate. We have modified our A/D algorithm to develop networks that tolerate multiple faults and degrade gracefully.

One possible direction for future research is to develop a hybrid algorithm of training and A/D to train and scale the size of the network dynamically, starting with a non-well-trained network. We can also compare the recovery speed (number of steps needed for retraining after faults occur) of networks $\mathcal{N}$ and $\mathcal{N}_{AD}$ when a tolerable number of faults occur.

Another possible approach to improve robustness is to build into the training algorithm a mechanism to discourage large weights: instead of the mean square error E_0, the new quantity to be minimized is of the form E_0+ (a penalty term monotonically increasing with the size of weights). We implemented three different possibilities, modifying each weight w_i by the quantities $-\eta(\frac{\partial E_0}{\partial w_i} + cw_i)$, $-\eta\frac{\partial E_0}{\partial w_i}(1 + cw_i)$, and $-\eta\left(\frac{\partial E_0}{\partial w_i}(1 + \frac{1}{2}c\sum_i w_i^2) + cw_iE_0\right)$, respectively, for many different values of c. This approach does improve robustness slightly, when compared to plain backpropagation, but the results are much less impressive than with our addition/deletion procedure.

We also performed experiments combining both approaches. The resulting improvements over the addition/deletion procedure were slight for Fisher's data and for the grid problem, but more pronounced for a character recognition data set (obtained from the database available from the Univ. of California at Irvine). Table 11 shows the result of robustness algorithm on two-letter character recognition data using the first modified learning rule mentioned above, changing each weight w_i by $-\eta(\frac{\partial E_0}{\partial w_i} + cw_i)$.

9 Fault Tolerance Enhancement for Neural Net Hardware

We have developed methods to adjust the weights of neural network such that all weights are within a limited range and still retain high robustness. The weights after adaptation can be directly used in a real neural network chip, such as Intel 80170NX ETANN, without reducing network performance. All inputs of networks are restricted from 0V to 2.8V, weights are restricted from -2.5V to 2.5V, and the built-in bias is always set to be 1V.

The learning rule used here is

$$\Delta w_i = -\eta(\frac{\partial E}{\partial w_i} + \lambda w_i) \tag{15}$$

which, in addition to the gradient descent term, has an extra term to penalize high magnitude

	Initial Net	Final Net
Hidden nodes	15	15
Training MSE	0.097931	0.099207
Testing MSE	0.105506	0.103928
Training Correctness(%)	86.20	86.40
Testing Correctness(%)	84.17	84.45
Avg. Sensitivity	0.025901	0.001453
Wst. Sensitivity	0.174443	0.013324

Table 11: Character Recognition using modified learning rule $\Delta w_i = -\eta(\frac{\partial E_0}{\partial w_i} + cw_i)$, with $c = 0.00005$. The deletion/addition process is $15 \rightarrow 3 \rightarrow 4 \rightarrow 3 \rightarrow 4 \rightarrow 5 \rightarrow 6 \rightarrow 7 \rightarrow 8 \rightarrow ... \rightarrow 15$.

weights[11]. Large weights will be generated only if they are really necessary. This will minimize the number of sensitive nodes, and the networks can be made more robust by reconfiguring with the Add/Delete procedure described in earlier reports.

Three methods are explored and tried on networks trained on Fisher's Iris data and a two-character recognition problem, injecting networks with many kinds of artificial faults. The first method is to clip the large weights which are out of range. The second method is to modify the training program such that all weights are trained within the limited range. The last approach is to design an algorithm which can convert a non-restricted network to a range-restricted network. We found that method three can retain the same performance and have better robustness than the original network since extra nodes are added systematically.

(1) Clipping: The simplest way to have all weights in a given range is to clip all weights of a well-trained neural network such that no weight is out of range. Since there is no retraining after clipping the weights, the performance of the clipped network is much worse than the original network.

(2) Training by Restricting Range of Weights:

To restrict the weights of a network to a given range, we modify the training algorithm by adding a restriction that a weight is allowed to be updated only if the weight will not thereby fall out of range. The learning rule is modified to be the following:

$$w_i(t+1) = \begin{cases} w_i(t) - \eta(\frac{\partial E}{\partial w_i} + \lambda w_i(t)) & \text{if } w_{min} \leq w_i(t) - \eta(\frac{\partial E}{\partial w_i} + \lambda w_i(t)) \leq w_{max} \\ w_{max} & \text{if } w_i(t) - \eta(\frac{\partial E}{\partial w_i} + \lambda w_i(t)) > w_{max}. \\ w_{min} & \text{if } w_i(t) - \eta(\frac{\partial E}{\partial w_i} + \lambda w_i(t)) < w_{min}. \end{cases}$$

where w_{min} and w_{max} are lower and upper bound of weights, respectively.

Training with restricted range of weights is similar to examining only a small portion of the error surface, and will converge to desired minima only if the minima happen to be

in this portion. From our experiments, it is difficult to reach a satisfactory minima using this training method unless the number of hidden nodes is increased until the error surface in the weight space of higher dimensionality contains the desired minima.

(3) Mapping Algorithm for Hardware Limitations

To adjust the parameters of a given network according to the physical limitations of Intel 80170 ETANN chip, we have developed a mapping algorithm called REFINE which reconfigures the given network to the hardware limitations.

In the proposed algorithm, there are two stages for adjustment of the weights of a neural network. First, we adjust the input-to-hidden layer weights by rescaling input samples and splitting input nodes, then we adjust the hidden-to-output layer weights by splitting hidden nodes. Since the bias input to all nodes (including hidden nodes and output nodes) is built-in with value 1, we have to avoid duplicating the threshold node. We made a small modification in the training procedure restricting all weights of bias links always to remain in the limited range.

To keep the same performance (same MSE) of the network after adjustment, we can keep the inner product of the input vector of each node and its associated weight vector unchanged. Suppose $\mathbf{X}$ is the original input vector and $\mathbf{W}$ is its associated weight vector, we would like to find a set of nodes with inputs $\mathbf{X}'_1, ..., \mathbf{X}'_n$ and weights $\mathbf{W}'_1, ..., \mathbf{W}'_n$ such that

$$\mathbf{X} \cdot \mathbf{W} = \sum_i \mathbf{X}'_i \cdot \mathbf{W}'_i,$$

where all elements in $\mathbf{X}'_i$ and $\mathbf{W}'_i$ are within the limited range. $\mathbf{X}'_i$ and $\mathbf{W}'_i$ can be found by rescaling $\mathbf{X}$ and $\mathbf{W}$, then expanding the size of vectors to reduce their magnitudes. Vector size expanding is implemented by adding extra nodes to reduce the magnitudes of the weights by "splitting".

The algorithm shown below is a mapping from a given network to a network with given physical limitations.

Notation: v_{ij} is the weight from input node i to hidden node j, w_{jk} is the weight from hidden node j to output node k, $x_i(p)$ is the i^{th} input of pattern p, the limited weight range is $[-w_{max}, +w_{max}]$, and the limited input range is $[0, x_{max}]$.

Step 1 All bias weights are trained with the restriction that they remain in the limited range.

Step 2 For some fan-out link of input node i, if the maximum absolute value of a weight is out of range, then rescale all weights associated with this node such that no weight is out of range. This will yield a factor α_i which will be multiplied by i^{th} column of input patterns. Let

$$\alpha_i = \begin{cases} \frac{\max_j\{|v_{ij}|\}}{w_{max}} & \text{if } \max_j\{|v_{ij}|\} > w_{max} \\ 1 & \text{otherwise.} \end{cases}$$

Then the weights and input patterns are rescaled to

$$v'_{ij} = \frac{v_{ij}}{\alpha_i}$$

$$x'_i(p) = x_i(p) \cdot \alpha_i$$

for each i (nodes and weights in input layer).

Step 3 For column i of input patterns, if the maximum absolute value of this column is out of range, then rescale this column such that no value of this column is out of range. This will also yield another factor β_i for multiplication with each weight associated to input node i. Let

$$\beta_i = \max(1, \frac{\max_p\{|x'_i(p)|\}}{|x_{max}|}).$$

The input patterns and weights are re-adjusted to

$$x''_i(p) = \frac{x'_i(p)}{\beta_i}$$

$$v''_{ij} = v'_{ij} \cdot \beta_i$$

for all nodes(i) in the input layer.

Step 4 For all fan-out links of input node i, if the maximum absolute value of weights is out of range, add extra nodes to split the weights of node i, and duplicate the i^{th} column of input such that the inner product of the input vector and weight vector is unchanged. Let

$$s_i = \begin{cases} \lceil \frac{\max_j\{|v''_{ij}|\}}{w_{max}} \rceil & \text{if } \max_j\{|v''_{ij}|\} > w_{max} \\ 1 & \text{otherwise} \end{cases}$$

If $s_i > 1$, the input node i is split into s_i nodes $\{n_i^0, n_i^1, ..., n_i^{s_i-1}\}$, and for each node n_i^q,$0 \leq q < s_i$,

$$v_{ij}^q = \frac{v''_{ij}}{s_i},$$

$$x_i^q(p) = x''_i(p).$$

Step 5 For hidden node j, if the maximum absolute value of fan-out links is out of range, then add extra nodes to split the weights in hidden-output layer and duplicate the weights in input-hidden layer. Let

$$t_j = \begin{cases} \lceil \frac{\max_k\{|w_{jk}|\}}{w_{max}} \rceil & \text{if } \max_k\{|w''_{jk}|\} > w_{max} \\ 1 & \text{otherwise} \end{cases}$$

If $t_j > 1$, the hidden node j is split into t_j nodes $\{n_j^0, n_j^1, ..., n_j^{t_j-1}\}$, and for each node n_j^q, $0 \leq q < t_j$,

$$w_{jk}^q = \frac{w_{jk}}{t_j},$$

$$v_{ij}^q = v_{ij}.$$

The purpose of **Step 2** and **Step 3** is to add as few extra nodes as possible. Usually, inputs are normalized between 0 to 1 which is a smaller range compare to our limitation $[0, 2.8]$, so there is some capacity here to help reduce magnitudes of weights between input and hidden layer. We make values of weights as small as possible without allowing input patterns to exceed their limitations. This will reduce the number of extra nodes to be duplicated. After this procedure, all the new input samples have to be pre-processed by rescaling and expanding, and all values larger than the maximum value of training samples have to be clipped since the rescaling factors are yielded via the maximum value of training samples. The new network is more robust than the original one because many extra nodes have been added in a systematic manner, preserving the input-output mapping of a well-trained network.

9.1 Experimental Results

We present experimental results on Fisher's Iris data and two-character recognition with different configurations of neural network on test data which are not presented in training samples. All the networks are tested on the following faults, and we assume that all faults occur on weights other than bias (threshold) values:

1. Single link faults. Perturb one link at a time by changing w_i to $w_i(1 + \alpha)$, where $-1 \leq \alpha \leq 1$.

2. Multiple link faults. Randomly inject simultaneous artificial faults to k links at a time for 1000 iterations, then find the average and worst case output errors.

3. Single link stuck at 0/1. Force one link weight to remain at 0 or 1 at a time. In our case, stuck at 0 is equal to set the weight to be -2.5, and stuck at 1 is equal to set the weight to be +2.5. Experiments are performed examining worst case and average case, with different links chosen for fault injection.

4. Single node stuck at 0/1. Force one node output to remain at 0 or 1 at a time. The node function used here is sigmoid $\frac{1}{1+\exp^{-h}}$.

For convenience, we use the following notation for each network:

1. N_{R1}: Trained by learning rule $R1$, given in equation 15.
2. $N_{R1/AD}$: Reconfigured by applying Add/Delete procedure to network N_{R1}.
3. N_{R1-FB}: Trained by $R1$ with the restriction that only bias weights are limited in the given range which is $[-2.5, 2.5]$ in our case.
4. $N_{R1-FB/AD}$: Reconfigured by applying Add/Delete procedure to N_{R1-FB}.
5. $N_{R1-FB/AD/RFN}$: $N_{R1-FB/AD}$ after applying REFINE procedure.

Only network $N_{R1-FB/AD/RFN}$ can be applied to Intel ETANN chip.

9.2 Fisher's Iris Data

Fisher's Iris data is for a three-class problem of classification and contains a four-dimensional input vector for each pattern. In building the neural networks, we rescaled the input data to fall between 0 and 1. There are 50 exemplars for each class. In our experiments we obtained a training set of size 100 consisting of the first 34, 35, and 31 exemplars of the three classes, respectively, and saved the remaining 50 exemplars to form the test set. All networks except $N_{R1-FB/AD/RFN}$ have 10 hidden nodes. $N_{R1-FB/AD/RFN}$ is generated from $N_{R1-FB/AD}$ by REFINE, and the number of hidden nodes increase to 40 after reconfiguration.

Although the network size is large, speed of computation is not affected because different hidden node computations are carried out in parallel, in the hardware implementation. Training occurs before the network size is expanded, hence training time is not affected; large network size hence does not mean that the learning algorithm is over-parameterized.

Figure 44 shows the comparison of different configurations with the first fault model, single-link faults. Different configurations are generated by Add/Delete procedure and REFINE after a 10 hidden-node network being trained by $R1$. Figure 45 shows the same comparison as figure 44 but on the second fault model, multiple links fault, with 10 link faults injected to network randomly at a time for 1000 trials. After applying REFINE procedure, network robustness improves significantly. Table 12 and 13 show the average and

	Network	Normal	Avg. Correctness		Wst. Correctness	
			stuck 0(-2.5)	stuck 1(+2.5)	stuck 0(-2.5)	stuck 1(+2.5)
1	N_{R1}	94%	91%	90%	58%	26%
2	$N_{R1/AD}$	94%	92%	91%	68%	30%
3	N_{R1-FB}	96%	90%	90%	58%	54%
4	$N_{R1-FB/AD}$	96%	90%	91%	64%	54%
5	$N_{R1-FB/AD/RFN}$	96%	95%	94%	82%	74%

Table 12: Average and worst case performances of networks with single-link stuck at 0/1 on Fisher's Iris data.

worst case on the third fault model, single link stuck at 0/1, and fourth fault model, single node stuck at 0/1, respectively. Again, the network $N_{R1/FB/AD/RFN}$ has the best robustness.

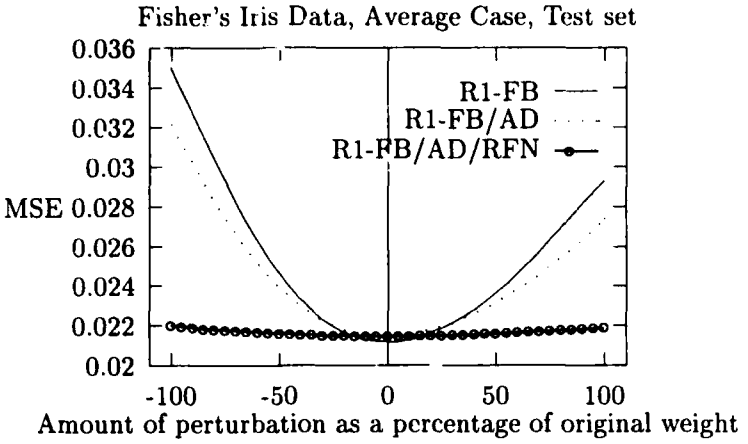
9.3 Two-Character Recognition

Letters "A" and "B" are selected from Letter Image Recognition Data created by David J. Slate in the UCI Repository Of Machine Learning Databases. We select 500, out of 1555, instances to form the training set, and leave the others for the test set. There are 262 instances of "A" and 238 instances of "B" in the training set. All networks have 15 hidden nodes. $N_{R1-FB/AD/RFN}$ is generated from $N_{R1-FB/AD}$ by REFINE, but no extra hidden nodes need to be added because the conditions of **Step 4** and **Step 5** are not satisfied.

Figure 46 and 47 show results on first and second fault models, respectively. $N_{R1/FB/AD}$ and $N_{R1/FB/AD/RFN}$ have the same robustness since no extra nodes are added after applying REFINE procedure. Table 14 and 15 show the average and worst cases on the third and fourth fault model, respectively.

9.4 Discussion

By merely modifying the weights that are out of range, we can neither retain the original performance nor improve the robustness. New training techniques to restrict weights, or mapping algorithms which reconfigure the network are necessary to develop robust networks that satisfy physical hardware constraints. In our method, we first use a training technique which restricts weights to the limited range during training. Since the weights are restricted during training, this network will need more hidden nodes than a non-restricted network. We have also designed a mapping procedure REFINE to change the configuration of a network to fit the limitation. The number of extra nodes needed depends on the magnitude of weights



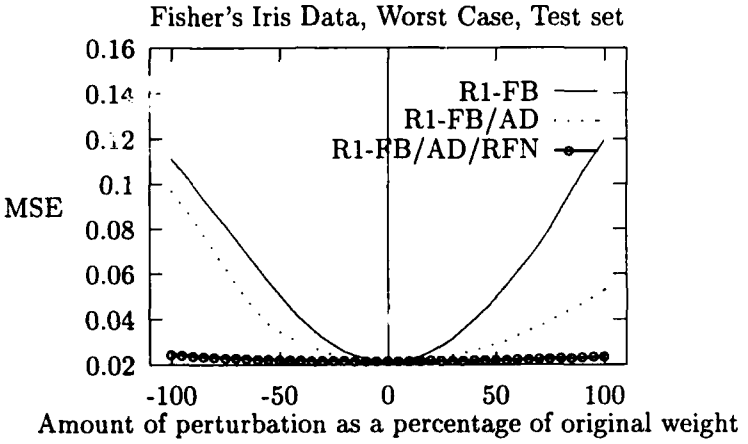
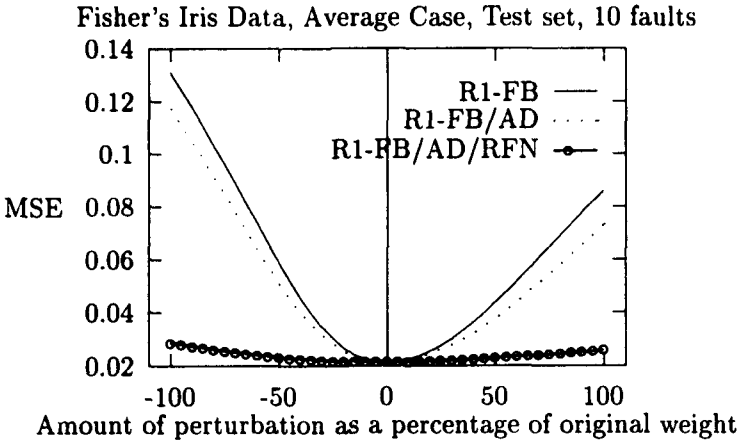


Figure 44: Comparison of different networks after applying Add/Delete procedure and REFINE with single-link fault on Fisher's Iris data.

	Network	Normal	Avg. Correctness		Wst. Correctness	
			stuck 0	stuck 1	stuck 0	stuck 1
1	N_{R1}	94%	82%	80%	66%	62%
2	$N_{R1/AD}$	94%	88%	88%	80%	72%
3	N_{R1-FB}	96%	77%	76%	62%	64%
4	$N_{R1-FB/AD}$	96%	80%	81%	64%	64%
5	$N_{R1-FB/AD/RFN}$	96%	95%	95%	94%	92%

Table 13: Average and worst case performances of networks with single-node stuck at 0/1 on Fisher's Iris data.



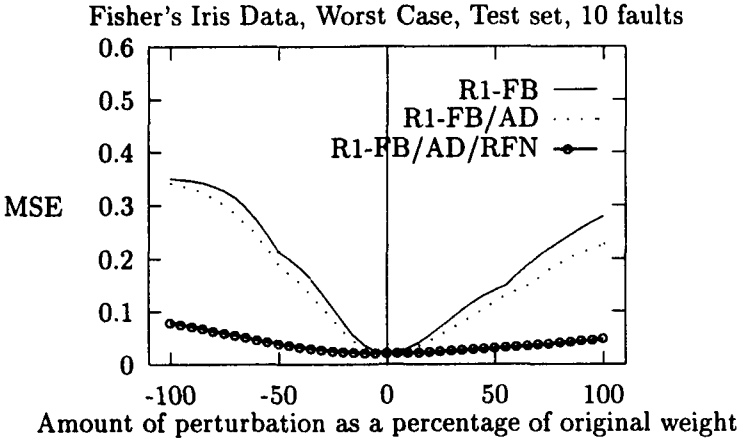
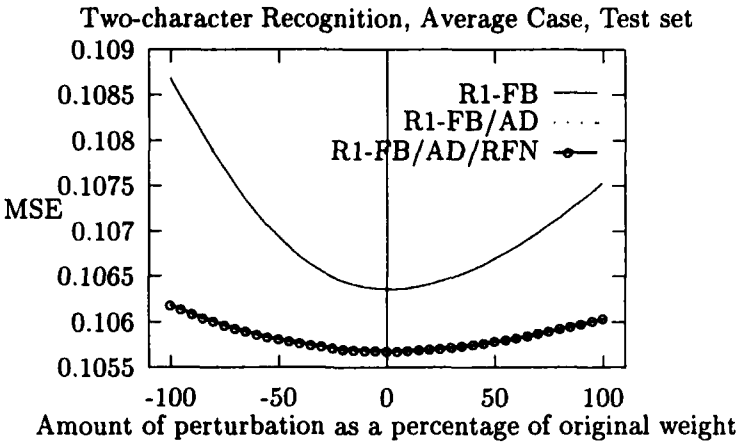


Figure 45: Comparison of different networks after applying Add/Delete procedure and REFINE with 10-link fault injected to network at a time for 1000 trials on Fisher's Iris data.

	Network	Normal	Avg. Correctness		Wst. Correctness	
			stuck 0(-2.5)	stuck 1(+2.5)	stuck 0(-2.5)	stuck 1(+2.5)
1	N_{R1}	85%	85%	85%	52%	65%
2	$N_{R1/AD}$	84%	84%	84%	60%	82%
3	N_{R1-FB}	85%	85%	85%	52%	65%
4	$N_{R1-FB/AD}$	85%	85%	84%	61%	82%
5	$N_{R1-FB/AD/RFN}$	85%	85%	85%	61%	81%

Table 14: Average and worst case performances of networks with single-link stuck at 0/1 on two-character recognition problem.



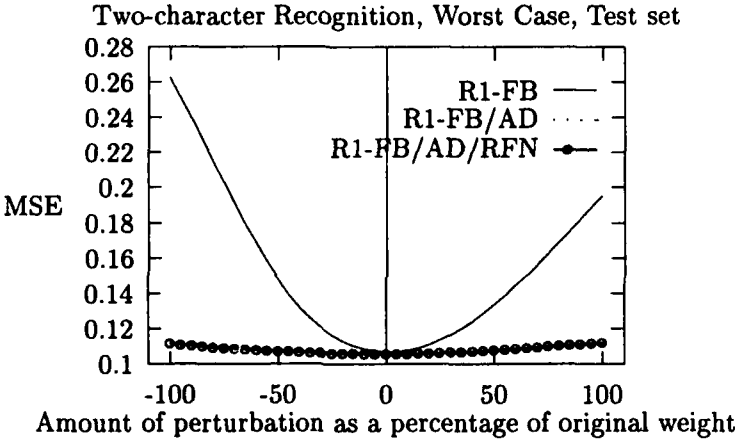


Figure 46: Comparison of different networks after applying Add/Delete procedure and REFINE with single-link fault on two-character recognition problem.

	Network	Normal	Avg. Correctness		Wst. Correctness	
			stuck 0	stuck 1	stuck 0	stuck 1
1	N_{R1}	85%	77%	77%	50%	50%
2	$N_{R1/AD}$	84%	79%	81%	69%	79%
3	N_{R1-FB}	85%	77%	77%	50%	50%
4	$N_{R1-FB/AD}$	85%	78%	81%	70%	78%
5	$N_{R1-FB/AD/RFN}$	85%	84%	81%	84%	78%

Table 15: Average and worst case performances of networks with single-node stuck at 0/1 on two-character recognition problem.

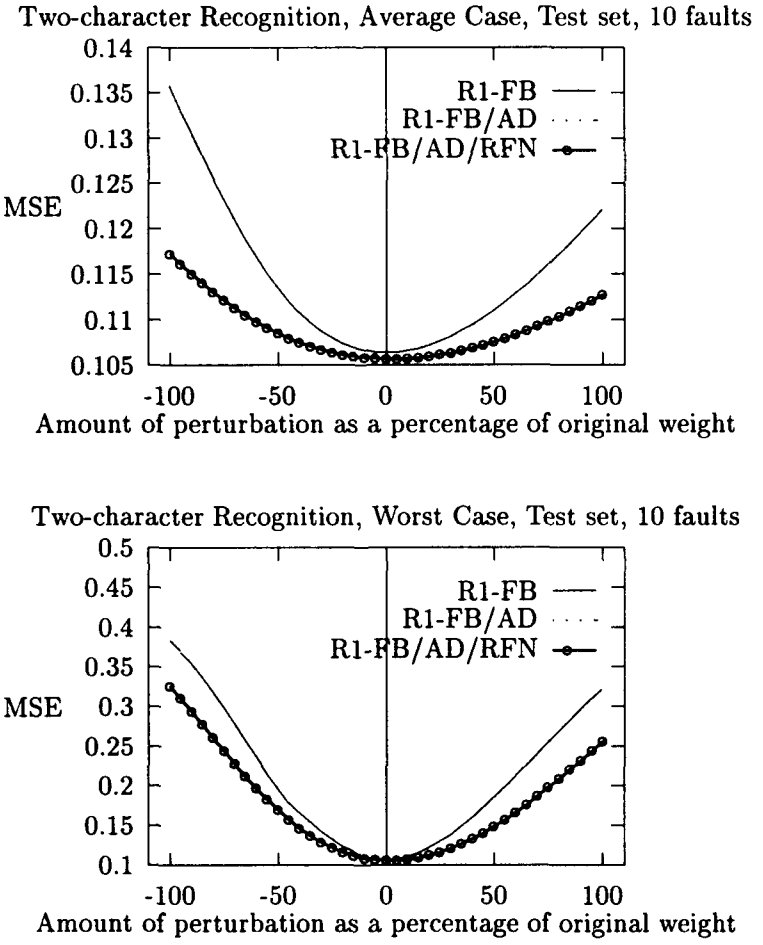


Figure 47: Comparison of different networks after applying Add/Delete procedure and REFINE with 10-link fault injected to network at a time for 1000 trials on two-character recognition problem.

and the number of spare resources available. We have tested these networks on four different fault models: single link perturbation, multiple link perturbation, single link stuck at-1/0 and single node stuck at-1/0. Experimental results showed that the robustness of the network is improved significantly after applying Add/Delete procedure and REFINE on these faults, and the network obtained by these methods can be directly applied to the Intel 80170NX ETANN chip which has hardware limitations.

10 Training by restricting weight magnitudes

In the previous section, we described an algorithm which can convert a non-restricted network to a range-restricted network such that all weights are within a limited range and still retain the robustness. The weights after adaptation can be applied to a real neural network chip, such as Intel 80170NX ETANN, without reducing the performance of the network.

The methods we developed in the previous reports to improve fault tolerance of neural networks are to adjust weights when training is completed. In this and the next section, two training techniques are developed to improve the fault tolerance of neural networks during the training process. One is to retain the low magnitude of weights during training and add hidden nodes dynamically to ensure desired performance can be reached. The other is to add artificial faults to the components of a network during training. The experimental results showed that both methods can obtain better robustness than backpropagation training.

Given a well-trained neural network, we found that the highly sensitive links have high magnitude weights in the network, but a link with high magnitude weight does not necessarily have high sensitivity. The importance of a link is not only determined by the weight it carries but also on the input it receives from the previous layer. If the input from the previous layer is small, the link has low sensitivity even if its weight has high magnitude. Restricting all weights to be small can guarantee that all links have low sensitivity.

10.1 Methodology and Algorithm

To obtain a more robust network, we should reduce the number of highly sensitive links without decreasing performance. Low magnitude weights can be retained by penalizing high magnitude weights during training. We have tried using R_1, which has an extra term to prevent high magnitude weights. However, the resulting weights using R_1 are either too large or too small because R_1 only discourages high weights but does not prohibit them, which means that high magnitude weights may be generated.

To ensure all weights are trained to be small, we modify the backpropagation training algorithm by limiting the magnitude of weights, such that the resulting weights are in a limited range with small magnitude. Since there is this extra restriction in the new learning

	fan_in	fan_out
n_s	w_i	$\frac{v_j}{2}$
n_{s1}	w_i	$\frac{v_j}{2}$
n_{s2}	Δw_i	v_j

Table 16: Weights re-distribution after adding two new hidden nodes.

rule, the network resulting from training may need more nodes than usual. To avoid selecting the number of hidden nodes in an *ad hoc* manner, we start with a small size network, then add new nodes to the network for further training if training converges. Convergence is detected by checking the amount of mean squared error changed in the most recent k steps; if it is smaller than some reference value, then we judge that the network has converged. Specifically, given a small value ϵ, the training is stopped if

$$\sum_{i=t-k+1}^{t} \Delta E(i) < \varepsilon,$$

where $\Delta E(i)$ is change in error in the i^{th} step and t is the current step.

When convergence is detected, two new nodes are added to the network to supplement the most sensitive node. After re-arranging the weight distribution, one node is added for continuously monitoring the $I - H$ layer weights, and the other is added to reduce the sensitivities of $H - O$ layer weights. Given the most sensitive node n_s with fan-in weights w_i and fan-out weights v_j, the two new nodes n_{s1} and n_{s2} are added with the weights re-distributed as shown in Table 16. Δw_i is the most recently computed increment for w_i, according to the generalized delta rule, irrespective of whether $w_i + \Delta w_i$ is out of range.

Given a one hidden layer feedforward network N with initial number of hidden nodes h_0, the Weight-Restricted Training Algorithm(WRTA) is shown below:

Step 1 Given desired error, ϵ, number of training iterations, t_{max}, upper and lower bound of weights w_{max} and w_{min}, and maximal number of hidden nodes to be added, h_{max}.

Step 2 $t = 0$, $h = h_0$.

Step 3

$$w_i(t+1) = \begin{cases} w_i(t) - \eta\frac{\partial E}{\partial w_i} & \text{if } w_{min} \le w_i(t) - \eta\frac{\partial E}{\partial w_i} \le w_{max} \\ w_{max} & \text{if } w_i(t) - \eta\frac{\partial E}{\partial w_i} > w_{max}. \\ w_{min} & \text{if } w_i(t) - \eta\frac{\partial E}{\partial w_i} < w_{min}. \end{cases}$$

Step 4 If $MSE(N) < \epsilon$ then STOP.

Step 5 If $\sum_{j=t-k+1}^{t} \Delta E(j) < \varepsilon$ and $h < h_{max}$ then
add two new nodes as described above, $h = h + 2$.

Step 6 $t = t + 1$, go to **Step 3**.

10.2 Experimental Results

1. The robustness of this new training technique is better than that of regular backpropagation, but many more training steps are required to achieve the same mean squared error. The sensitivity of each node is roughly equal at the end of applying this algorithm, which means all nodes are equally important.

2. The Add/Delete procedure with no hardware restrictions constructs more robust networks than the new training technique, but cannot be used for transferring the trained network to hardware.

3. MSE falls more slowly during training, with this new algorithm.

11 Trained by Injecting Artificial Faults

Clay and Sequin[8] developed a training technique to improve the fault tolerance of neural networks by randomly setting the outputs of some hidden nodes to be zero in each iteration during training. Bolt[6] use the similar idea but setting the weights of links to be zero instead of the outputs of hidden nodes. From our experiments on Clay and Bolt's methods for improving fault tolerance, we found that injecting a specific fault to a network during the training process can evolve a network which can tolerate that specific fault. In Clay's method, for example, hidden units are disabled by setting the outputs of those units to zero, which improved fault tolerance when the fault is to stick the output at zero. But a network trained using this method does not tolerate other kinds of faults, e.g., when the fault is to stick the output at one, or to some other perturbation. A similar situation occured in Bolt's method which trains the network with randomly disabled weights.

11.1 Methodology

Based on these concepts of training to improve fault tolerance, we developed a method that injects different types of faults into a network during the training process to produce a set of weights that is more robust against various types of faults. Specifically, in each iteration of the training process, we randomly choose a fixed number of hidden nodes, then set the outputs of these nodes to be zero and one alternately. In the same iteration, a small number of links are also selected randomly to be perturbed. When all the artificial faults are

Network	Normal	Avg. Correctness	Wst. Correctness
BP	99%	94%	65%
Clay(4)	98%	97%	92%
Bolt(5)	99%	96%	72%
Comb1(4-5)	98%	97%	91%
Comb2(4-5)	97%	95%	77%
A/D	99%	98%	96%

Table 17: Average and worst case performances of networks with single-link stuck at 0 on Fisher's Iris training data.

Network	Normal	Avg. Correctness	Wst. Correctness
BP	94%	90%	58%
Clay(4)	96%	95%	92%
Bolt(5)	92%	91%	72%
Comb1(4-5)	96%	95%	90%
Comb2(4-5)	94%	92%	76%
A/D	96%	94%	90%

Table 18: Average and worst case performances of networks with single-link stuck at 0 on Fisher's Iris test data.

injected, feedforward computations are performed to obtain the delta errors of each weight, then weights are updated with these delta errors. This is a combination of Clay and Bolt's methods.

11.2 Experimental Results

We compare backpropagation, Clay's method, Bolt's method, and Add/Delete procedure on Fisher's Iris data and the four-class grid discrimination problem; the results are shown in the tables that follow. The best performance is obtained by our A/D algorithm for worst case analysis, on almost every kind of fault. The next best results are obtained for the new combination algorithms Comb1 and Comb2 that we have synthesized. Plain backpropagation does worse than every other method, and Clay and Bolt's methods show intermediate performance.

Network	Normal	Avg. Correctness		Wst. Correctness	
		stuck 0	stuck 1	stuck 0	stuck 1
BP	99%	83%	86%	63%	48%
Clay(4)	98%	98%	69%	98%	61%
Bolt(5)	99%	89%	82%	66%	55%
Comb1(4-5)	98%	97%	76%	92%	56%
Comb2(4-5)	97%	93%	87%	79%	74%
A/D	99%	96%	95%	91%	93%

Table 19: Average and worst case performances of networks with single-node stuck at 0/1 on Fisher's Iris training data.

Network	Normal	Avg. Correctness		Wst. Correctness	
		stuck 0	stuck 1	stuck 0	stuck 1
BP	94%	80%	86%	62%	50%
Clay(4)	96%	94%	72%	92%	64%
Bolt(5)	92%	87%	82%	70%	58%
Comb1(4-5)	96%	95%	79%	94%	62%
Comb2(4-5)	94%	91%	86%	78%	74%
A/D	96%	94%	93%	88%	90%

Table 20: Average and worst case performances of networks with single-node stuck at 0/1 on Fisher's Iris test data.

Network	Normal	Avg. Correctness	Wst. Correctness
BP	96%	88%	49%
Clay(1)	95%	90%	59%
Bolt(3)	96%	92%	65%
Comb1(1-5)	96%	94%	78%
Comb2(2-5)	96%	95%	91%
A/D	95%	93%	90%

Table 21: Average and worst case performances of networks with single-link stuck at 0 on Grid Discrimination training data.

Network	Normal	Avg. Correctness	Wst Correctness
BP	95%	87%	49%
Clay(1)	94%	89%	57%
Bolt(3)	95%	91%	63%
Comb1(1-5)	96%	94%	76%
Comb2(2-5)	94%	93%	90%
A/D	96%	94%	87%

Table 22: Average and worst case performances of networks with single-link stuck at 0 on Grid Discrimination test data.

Network	Normal	Avg. Correctness		Wst. Correctness	
		stuck 0	stuck 1	stuck 0	stuck 1
BP	96%	79%	77%	49%	49%
Clay(1)	95%	89%	75%	82%	59%
Bolt(3)	96%	86%	81%	70%	65%
Comb1(1-5)	96%	92%	87%	85%	78%
Comb2(2-5)	96%	93%	94%	90%	92%
A/D	95%	90%	90%	90%	90%

Table 23: Average and worst case performances of networks with single-node stuck at 0/1 on Grid Discrimination training data.

Network	Normal	Avg. Correctness		Wst. Correctness	
		stuck 0	stuck 1	stuck 0	stuck 1
BP	94%	78%	77%	49%	48%
Clay(1)	94%	88%	75%	83%	57%
Bolt(3)	95%	85%	81%	72%	63%
Comb1(1-5)	96%	92%	87%	85%	76%
Comb2(2-5)	94%	92%	92%	89%	90%
A/D	96%	92%	88%	92%	87%

Table 24: Average and worst case performances of networks with single-node stuck at 0/1 on Grid Discrimination test data.

APPENDIX (Proofs for theorem in Section 8) The statement of the theorem as stated in Section 8 does not precisely mention the sensitivity measure used; in this appendix, proofs are presented for various propositions that are different cases of the theorem for different definitions of sensitivity.

Notation: Let $E(N^{\alpha}_{mn})$ denote the error obtained when the link weight ν_{mn} in the network N is replaced by $(1+\alpha)\nu_{mn}$. Similarly, let $E(N^{\alpha}_{k})$ denote the average error obtained when each of the link weights ν_{mk} in the network N is replaced by $(1+\alpha)\nu_{mk}$. We remind that ν_{ij} denotes the weight on the link from the j^{th} hidden node to the i^{th} output node.

Proposition 1 *Let N be a well-trained[7] I-h-O network in which link ν_{ij} is more sensitive than every other link in the second (ν) layer, where sensitivity is defined as (cf. Equations 3, 4) the additional error resulting from perturbation of any ν_{mn} to $(1+\alpha)\nu_{mn}$, for some α (i.e., with the singleton perturbation set $A = \{\alpha\}$) such that these perturbations degrade performance, (i.e., the error $E(N) < \max_{m,n}\{E(N^{\alpha}_{mn})\}$.*

Let M be the network obtained by adding a redundant $(h+1)^{st}$ hidden node to M and adjusting weights of links attached to this new node and to the j^{th} hidden node, as specified in the ADP given earlier.

Then M is more robust than N, i.e., the sensitivity of M is lower than that of N.

Proof: M will be more robust than N iff $max_{m,n}\{E(< N^{\alpha}_{mn})\} > max_{m,n}\{E(M^{\alpha}_{mn})\}$. Since ν_{ij} is the most sensitive node in N, we need to show that $E(N^{\alpha}_{ij}) > max_{m,n}\{E(M^{\alpha}_{mn})\}$. There are two cases, depending on whether the link weight ν_{mn} was affected by the addition of the $h+1^{st}$ node (by modifying N to M).

Case 1: Links connected to unduplicated hidden nodes.

Since ν_{ij} is the most sensitive node in N, $E(N^{\alpha}_{ij}) > E(N^{\alpha}_{mk})$, $\forall k.\ k \neq j \ \ \& \ \ k \neq h+1$. Since the output behaviors of M and N are identical (when neither is perturbed), and since other links are left undisturbed, an identical change in error occurs in M and N when there is a perturbation in any link other than those connected to the j^{th} or the $h+1^{st}$ hidden nodes. In other words, $\forall k.\ k \neq j \& k \neq h+1$, and $\forall m$, $E(N^{\alpha}_{mk}) = E(M^{\alpha}_{mk})$. Therefore $E(N^{\alpha}_{ij}) > E(M^{\alpha}_{mk})$, $\forall k.\ k \neq j \ \ \& \ \ k \neq h+1$.

Case 2: Links connected to j^{th} and $h+1^{st}$ hidden nodes. Then

$$0 \leq E(N) \leq E(N^{\alpha}_{mj}) < E(N^{\alpha}_{ij}),$$

by the premises of the Proposition. If the target value for the l^{th} output node is t_l, and 'S_k' abbreviates a sigmoid node function so that $S_k(x) = e^{x+c_k}/(1+e^{x+c_k}))$, where c_k abbreviates other terms contributing to the k^{th} node's output, then

$$0 \leq \quad \|(S_i(\nu_{ij}y_j) - t_i)\| + \|(S_m(\nu_{mj}y_j) - t_m)\|$$

[7] "Well-trained" means that the network error is almost 0.

$$\leq \|(S_i(\nu_{ij}y_j) - t_i)\| + \|(S_m(\nu_{mj}y_j(1+\alpha)) - t_m)\|$$
$$< \|(S_i(\nu_{ij}y_j(1+\alpha)) - t_i)\| + \|(S_m(\nu_{mj}y_j) - t_m)\|.$$

This implies that $\nu_{mj}\alpha \geq 0$ if $t_m \approx 0$, and $\nu_{mj}\alpha \leq 0$ if $t_m \approx 1$. Also, irrespective of t_m,

$$\|(S_i(\nu_{ij}y_j(1+\alpha))-t_i)\|+\|(S_m(\nu_{mj}y_j)-t_m)\|-\|(S_i(\nu_{ij}y_j)-t_i)\|-\|(S_m(\nu_{mj}y_j(1+\alpha))-t_m)\| > 0 \quad (1)$$

Let $n = j$ or $n = h + 1$, so that ν_{mn} in M equals $\nu_{mj}/2$ in N, and $y_n = y_j$ in both M and N. Then

$E(N_{ij}^{\alpha}) - E(M_{mn}^{\alpha})$

$= \|(S_i(\nu_{ij}y_j(1+\alpha))-t_i)\|+\|(S_m(\nu_{mj}y_j)-t_m)\|-\|(S_i(\nu_{ij}y_j)-t_i)\|-\|(S_m(\nu_{mj}y_j(1+\alpha/2))-t_m)\|$

$= \|(S_i(\nu_{ij}y_j(1+\alpha))-t_i)\|+\|(S_m(\nu_{mj}y_j)-t_m)\|-\|(S_i(\nu_{ij}y_j)-t_i)\|-\|(S_m(\nu_{mj}y_j(1+\alpha))-t_m)\|$

$$+\|(S_m(\nu_{mj}y_j(1+\alpha)) - t_m)\| - \|(S_m(\nu_{mj}y_j(1+\alpha/2)) - t_m)\|$$

$> \|(S_m(\nu_{mj}y_j(1+\alpha))-t_m)\|-\|(S_m(\nu_{mj}y_j(1+\alpha/2))-t_m)\|$. If $t_m \approx 0$, then this last quantity is positive because $\nu_{mj}\alpha \geq 0$ and hence $\nu_{mj}y(1+\alpha)-\nu_{mj}y_j(1+\alpha/2) \leq 0$, given that y_j is always between 0 and 1, and the sigmoid S_m is a mapping into the interval [0,1]. Similarly, if $t_m \approx 1$, then this quantity is positive because $\nu_{mj}\alpha \leq 0$, hence $\nu_{mj}y(1+\alpha) - \nu_{mj}y_j(1+\alpha/2) \leq 0$.

□

One of the premises of the above proposition is that perturbations should degrade performance[8]: if such is not the case, i.e., if network error actually decreases as a result of introducing "faults" into the system, then our algorithm replaces the network by the new 'perturbed' network with better performance, and retrains that network.

The above proposition pertained to the special case where A was a singleton set. If A contains more elements, two definitions for link sensitivity are possible: one which considers the average degradation of links (averaged over different perturbations $\in A$), and another which considers the worst case degradation of links (among different perturbations $\in A$). The above Proposition generalizes in both cases, with almost identical proofs, when α is chosen to be the maximal element of A.

Proposition 2 *Let N be a well-trained I-h-O network in which link ν_{ij} is more sensitive than every other link in the second layer, where sensitivity is defined as increase in error caused by perturbation of any ν_{mn} to $\max_{\alpha_k \in A}\{(1+\alpha_k)\nu_{mn}\}$, for some A such that these perturbations degrade performance, i.e., the error $E(N) < max_{m,n}\{E(N_{mn}^{\alpha})\}$, $\forall \alpha \in A$.*

[8] A simple network can be constructed which does not obey this criterion, and hence splitting the node connected to the most sensitive edge and perturbing edges does not yield as good an improvement of MSE as perturbing the link in the original network.

Let M be the network obtained by adding a redundant $(h+1)^{st}$ hidden node to M and adjusting weights of links attached to this new node and to the j^{th} hidden node, as specified in the ADP given earlier. Then M is more robust than N, i.e., the sensitivity of M is lower than that of N.

Proof: M will be more robust than N iff $max_{m,n}\{E(N_{mn}^{\alpha_k})\} > max_{m,n}\{E(M_{mn}^{\alpha_l})\}\ \forall \alpha_k, \alpha_l \in A$. Let $\alpha \in A$ be the perturbation in ν_{ij} causing most error. Since ν_{ij} is the most sensitive node, we need to show that $E(N_{ij}^{\alpha}) < max_{m,n}\{E(M_{mn}^{\alpha_k})\}$ for every $\alpha_k \in A$.

Case 1: Links connected to unduplicated hidden nodes.

As in the above Proposition, an identical change in error occurs in M and N when there is a perturbation in any link other than those connected to the j^{th} or the $h+1^{st}$ hidden nodes. So $\forall k.\ k \neq j \& k \neq h+1$, $\forall \alpha_l \in A$, and $\forall m$, $E(N_{mk}^{\alpha_l}) = E(M_{mk}^{\alpha_l})$ and $E(N_{ij}^{\alpha}) < E(N_{mk}^{\alpha_l}) = E(M_{mk}^{\alpha_l})$.

Case 2: Links connected to j^{th} and $h+1^{st}$ hidden nodes.

As in the preceding Proposition,

$$0 \leq E(N) \leq E(N_{mj}^{\alpha_l}) < E(N_{ij}^{\alpha}),$$

for any $\alpha_l \in A$, and

$$\begin{aligned} 0 \leq \quad & \|(S_i(\nu_{ij}y_j) - t_i)\| + \|(S_m(\nu_{mj}y_j) - t_m)\| \\ \leq\ & \|(S_i(\nu_{ij}y_j) - t_i)\| + \|(S_m(\nu_{mj}y_j(1+\alpha_l)) - t_m)\| \\ <\ & \|(S_i(\nu_{ij}y_j(1+\alpha)) - t_i)\| + \|(S_m(\nu_{mj}y_j) - t_m)\|. \end{aligned}$$

This implies that $\nu_{mj}\alpha_l \geq 0$ if $t_m \approx 0$, and $\nu_{mj}\alpha_l \leq 0$ if $t_m \approx 1$. Again, irrespective of t_m,

$$\|(S_i(\nu_{ij}y_j(1+\alpha))-t_i)\|+\|(S_m(\nu_{mj}y_j)-t_m)\|-\|(S_i(\nu_{ij}y_j)-t_i)\|-\|(S_m(\nu_{mj}y_j(1+\alpha_l))-t_m)\| > 0$$

If $n = j$ or $n = h+1$, then

$$\begin{aligned} & E(N_{ij}^{\alpha}) - E(M_{mn}^{\alpha_l}) \\ &= \|S_i(\nu_{ij}y_j(1+\alpha)) - t_i\| + \|S_m(\nu_{mj}y_j) - t_m\| - \|S_i(\nu_{ij}y_j) - t_i\| - \|S_m(\nu_{mj}y_j(1+\alpha_l/2)) - t_m\| \\ &= \|S_i(\nu_{ij}y_j(1+\alpha)) - t_i\| + \|S_m(\nu_{mj}y_j) - t_m\| - \|S_i(\nu_{ij}y_j) - t_i\| - \|S_m(\nu_{mj}y_j(1+\alpha_l)) - t_m\| \end{aligned}$$

$$+\|S_m(\nu_{mj}y_j(1+\alpha_l)) - t_m\| - \|S_m(\nu_{mj}y_j(1+\alpha_l/2)) - t_m\|$$

$> \|(S_m(\nu_{mj}y_j(1+\alpha_l)) - t_m)\| - \|(S_m(\nu_{mj}y_j(1+\alpha_l/2)) - t_m)\|$. If $t_m \approx 0$, then this quantity is positive because $\nu_{mj}\alpha_l \geq 0$ and hence $\nu_{mj}y(1+\alpha_l) - \nu_{mj}y_j(1+\alpha_l/2) \leq 0$. Similarly, if $t_m \approx 1$, then this quantity is positive because $\nu_{mj}\alpha_l \leq 0$, hence $\nu_{mj}y(1+\alpha_l) - \nu_{mj}y_j(1+\alpha_l/2) \leq 0$.

Since the above holds for any $\alpha_l \in A$, we have $E(N_{ij}^{\alpha}) > \max_{\alpha_l \in A, \& \forall m} E(M_{mn}^{\alpha_l})\}$

□

Proposition 3 *Let N be a well-trained I-h-O network in which link ν_{ij} is more sensitive than every other link in the second layer, where sensitivity is defined as increase in error caused by perturbation of any ν_{mn} to $\frac{1}{|A|}\sum_{\alpha_k \in A}\{(1+\alpha_k)\nu_{mn}\}$, for some A such that these perturbations degrade performance, i.e., the error $E(N) < max_{m,n}\{E(N^{\alpha}_{mn})\}$, $\forall \alpha \in A$.*

Let M be the network obtained by adding a redundant $(h+1)^{st}$ hidden node to M and adjusting weights of links attached to this new node and to the j^{th} hidden node, as specified in the ADP given earlier.

Then M is more robust than N, i.e., the sensitivity of M is lower than that of N.

Proof: As before, M will be more robust than N iff

$$\sum_{\alpha \in A}\{E(N^{\alpha}_{mn})\} > max_{m,n}\sum_{\alpha \in A}\{E(M^{\alpha}_{mn})\}.$$

Since ν_{ij} is the most sensitive node, we need to show that

$$\sum_{\alpha \in A} E(N^{\alpha}_{ij}) < max_{m,n}\sum_{\alpha \in A}\{E(M^{\alpha}_{mn})\}.$$

Case 1: Links connected to unduplicated hidden nodes.

As in the above Proposition, an identical change in error occurs in M and N when there is a perturbation in any link other than those connected to the j^{th} or the $h+1^{st}$ hidden nodes. So $\forall k.\ k \neq j \& k \neq h+1$, $\forall \alpha_l \in A$, and $\forall m$, $E(N^{\alpha_l}_{mk}) = E(M^{\alpha_l}_{mk})$ and $E(N^{\alpha}_{ij}) < E(N^{\alpha_l}_{mk}) = E(M^{\alpha_l}_{mk})$.

Case 2: Links connected to j^{th} and $h+1^{st}$ hidden nodes.

As in the preceding Proposition,

$$0 \leq E(N) \leq E(N^{\alpha}_{mj})$$

for any $\alpha \in A$. Since ν_{ij} is the most sensitive link,

$$\sum_{\alpha \in a} E(N^{\alpha}_{mj}) < \sum_{\alpha \in a} E(N^{c}_{ij}).$$

As before, if $n = j$ or $n = h+1$, then

$\sum_{\alpha \in a}[E(N^{\alpha}_{ij}) - E(M^{\alpha}_{mn})]$

$= \sum_{\alpha \in a}[||(S_i(\nu_{ij}y_j(1+\alpha)) - t_i)|| + ||(S_m(\nu_{mj}y_j) - t_m)|| - ||(S_i(\nu_{ij}y_j) - t_i)|| - ||(S_m(\nu_{mj}y_j(1+\alpha/2)) - t_m)||]$

$= \sum_{\alpha \in a}[||(S_i(\nu_{ij}y_j(1+\alpha)) - t_i)|| + ||(S_m(\nu_{mj}y_j) - t_m)|| - ||(S_i(\nu_{ij}y_j) - t_i)|| - ||(S_m(\nu_{mj}y_j(1+\alpha)) - t_m)||]$

$$+ \sum_{\alpha \in a}[||(S_m(\nu_{mj}y_j(1+\alpha)) - t_m)|| - ||(S_m(\nu_{mj}y_j(1+\alpha/2)) - t_m)||]$$

$> \sum_{\alpha \in a}[||(S_m(\nu_{mj}y_j(1+\alpha)) - t_m)|| - ||(S_m(\nu_{mj}y_j(1+\alpha/2)) - t_m)||]$

Again, if $t_m \approx 0$, then each quantity in the summation is positive because $\nu_{mj}\alpha \geq 0$ and hence $\nu_{mj}y(1+\alpha) - \nu_{mj}y_j(1+\alpha/2) \leq 0$. Similarly, if $t_m \approx 1$, then each summed quantity is positive because $\nu_{mj}\alpha \leq 0$, hence $\nu_{mj}y(1+\alpha) - \nu_{mj}y_j(1+\alpha/2) \leq 0$.

□

We turn our attention now to node sensitivity. Node sensitivity may be defined as the error resulting from perturbing the node's output, rather than perturbing the weights of adjacent links (Equation 5). However, since every hidden node has the same outdegree (=O, the number of output nodes), the error obtained by perturbing a hidden node's output is a constant multiple (O) of the average error obtained by perturbing the links outgoing from that hidden node.

Proposition 4 *Let N be a well-trained I-h-O network in which the i^th hidden node is more sensitive than every other hidden node, where sensitivity is defined as error degradation caused by perturbation of links ν_{mn} to $(1+\alpha)\nu_{mn}$, for some α, such that these perturbations degrade performance.*

Let M be the network obtained by adding a redundant $(h+1)^{st}$ hidden node to M and adjusting weights of links attached to this new node and to the j^{th} hidden node, as specified in the ADP given earlier.

Then M is more robust than N, i.e., the sensitivity of M is lower than that of N.

Proof: Let the k^{th} hidden node in M be the most sensitive.

As in the previous propositions, if $k \neq j$ and $k \neq h+1$, then $0 \leq E(N_k^\alpha) = E(M_k^\alpha) < E(N_j^\alpha)$, hence M is less sensitive than N.

Now consider the case where if $k = j$ or $k = h+1$. We need to show that $E(N_j^\alpha) > E(M_k^\alpha)$, i.e., $E(N_j^\alpha) > E(M_j^\alpha)$, i.e.,

$$\sum_m [\|S_m(\nu_{mj}(1+\alpha)y_j) - t_m\| - \|S_m(\nu_{mj}(1+\alpha/2)y_j) - t_m\|] > 0.$$

The term in the summation is positive for each m, because of the premise that $0 \leq E(N) \leq E(N_j^\alpha)$, since, as in the previous propositions,

$$t_m \approx 0 \;\Rightarrow\; \nu_{mj}\alpha > 0 \;\Rightarrow\; \nu_{mj}\alpha > \nu_{mj}\alpha/2, \text{ and}$$

$$t_m \approx 1 \;\Rightarrow\; \nu_{mj}\alpha < 0 \;\Rightarrow\; \|S_m(\nu_{mj}(1+\alpha)y_j) - t_m\| > \|S_m(\nu_{mj}(1+\alpha/2)y_j) - t_m\|$$

□

Proposition 5 *Let N be a well-trained I-h-O network in which link w_{ij} is more sensitive than every other link in the first layer of weights, where sensitivity is defined as increase in error caused by perturbation of any w_{mn} to $(1+\alpha)w_{mn}$, for some α such that these perturbations degrade performance.*

Let M be the network obtained by adding a redundant $(h+1)^{st}$ hidden node to M and adjusting weights of links attached to this new node and to the j^{th} hidden node, as specified in the ADP given earlier.

Then M is more robust than N, i.e., the sensitivity of M is lower than that of N.

Proof: The perturbation of one first layer weight w_{ij} is precisely equivalent to perturbing the relevant (i^{th}) hidden node's output by some quantity α_i. Hence the result follows from the previous proposition: perturbations in other hidden node outputs cause less network error than the most sensitive hidden node's perturbation, and halving the outgoing link weights from that node decreases the effect of perturbation of that hidden node.

□

As in the case of propositions 2 and 3, it can be shown that network robustness is increased by our node duplication algorithm even for multiple first layer weight changes, changes by a non-singleton set A of perturbations, multiple node changes, and perturbations in any fixed number of multiple links.

Proposition 6 *Let N be a well-trained I-h-O network in which link ν_{ij} is more sensitive than every other link in the second layer, where sensitivity is defined as perturbation of any ν_{mn} to $\nu_{mn}+\Delta$, for some Δ such that these perturbations degrade performance, i.e., the error $E(N) < E(N^{\Delta}_{mn})\ \forall \nu_{mn}$, where $E(N^{\Delta}_{mn})$ is the error obtained when the link ν_{mn} is replaced by $\nu_{mn}+\Delta$.*

Let N' be the network obtained by adding a redundant $(h+1)^{st}$ hidden node to N and adjusting weights as specified in the ADP given earlier.

Then N' is more robust than N, i.e., the sensitivity of N' is lower than that of N.

Proof: Similar to the preceding proposition.

□

References

[1] R.Anand, K.Mehrotra, C.K.Mohan, S.Ranka,*Analyzing Images Containing Multiple Sparse Patterns using Neural Networks,* submitted to Int'l. Joint Conf. on Artificial Intelligence, 1991.

[2] T.Anderson and P.A.Lee, Fault Tolerance: Principles and Practice, Prentice-Hall, 1981.

[3] N.Asokan, R.Shankar, K.Mehrotra, C.K.Mohan, S.Ranka, *A Neural Network Simulator on the Connection Machine,* in Proc. IEEE Int'l. Symp. on Intelligent Control, Sept. 1990.

[4] D.Ballard, *Modular Learning in Hierarchical Neural Networks*, in Computational Neuroscience (ed. E.L.Schwartz), MIT Press, 1990, 139-153.

[5] L.A.Belfore, Fault Tolerance of Neural Networks, Ph.D. Dissertation, Univ. of Virginia, 1989.

[6] G. Bolt, *Investigating Fault Tolerance in Artificial Neural Networks* University of York, Department of Computer Science, Technical Report YCS 154, Heslington, York, England, 1991

[7] M.J. Carter, F.J.Rudolph and A.J. Nucci, *Operational Fault Tolerance of CMAC Networks*, Proc. Workshop on Neural Info. Proc. Systems, 1989, 340-347.

[8] R.D. Clay and C.H. Sequin, *Limiting Fault-Induced Output Errors in ANN's*, Proc. Int'l. Joint Conf. Neural Networks, Seattle, 1990, vol.I:337-340.

[9] W.H.Debany, Jr., and C.R.P.Hartmann, *Testability Measurement: A Systematic Approach*, Proc. Government Microcircuit Application Conf. (GOMAC), San Diego, Nov. 1986.

[10] D.E.Goldberg, Genetic Algorithms in Search, Optimization, and Machine Learning, Addison-Wesley Pub. Co., 1989.

[11] A. Krogh, J.A. Hertz, *A Simple Weight Decay Can Improve Generation.*

[12] B.W.Johnson, Design and Analysis of Fault Tolerant Digital Systems, Addison-Wesley, 1989.

[13] E.D. Karnin, *A Simple Procedure for Pruning Back-Propagation Trained Neural Networks*, IEEE Trans. Neural Networks 1 (2), June 1990, 239-242.

[14] J.Koh, G.S.Moon, K.Mehrotra, C.K.Mohan and S.Ranka, *Korean Character Recognition using Neural Networks,* in Proc. Conf. on Frontiers of Massively Parallel Computation, Oct. 1990.

[15] P.K.Lala, Fault Tolerant and Fault Testable Hardware Design, Prentice-Hall, 1985.

[16] M.Li, K.Mehrotra, C.K.Mohan, S.Ranka, *Forecasting Sunspot Numbers using Neural Networks,* in Proc. IEEE Int'l. Symp. on Intelligent Control, Sept. 1990.

[17] K.G.Mehrotra, C.K.Mohan, S.Ranka, *Bounds on the Number of Samples Needed for Neural Learning,* submitted to *IEEE Transactions on Neural Networks.*

[18] V.P.Nelson and B.D.Carroll, Fault-tolerant Computing: Tutorial, IEEE Computer Science Press, 1987.

[19] D.N.Osherson, S.Weinstein and M.Stob, *Modular Learning,* in Computational Neuroscience (ed. E.L.Schwartz), MIT Press, 1990, 369-377.

[20] T. Petsche and B.W. Dickinson, *Trellis Codes, Receptive Fields, and Fault Tolerant, Self-Repairing Neural Networks*, IEEE Trans. Neural Networks 1 (2), June 1990, 154-166.

[21] P.W. Protzel and M.K. Arras, *Fault Tolerance of Optimization Networks: Treating Faults as Additional Constraints*, Proc. Int'l. Joint Conf. Neural Networks, San Diego, June 1990, vol.I:455-458.

[22] D.E.Rumelhart, J.L.McClelland, and the PDP Research Group, *Parallel Distributed Processing,* Vol.1, 1986.

[23] M.Stevenson, R.Winter and B.Widrow, *Sensitivity of Layered Neural Networks to Errors in the Weights*, Proc. Int'l. Joint Conf. Neural Networks, San Diego, June 1990, vol.I:337-340.

[24] G. Swaminathan, S. Srinivasan and S. Mitra, *Fault Tolerance in Neural Networks*, Proc. Int'l. Joint Conf. Neural Networks, San Diego, June 1990, vol.II:699-702.

[25] S.S.Venkatesh, *Robustness in Neural Computation: Random Graphs and Sparsity*, Technical Report, Univ. of Pennsylvania, Dec. 1990.

[26] D.Zipser, *Subgrouping Reduces Complexity and Speeds Up Learning in Recurrent Networks*, in Advances in Neural Information Processing Systems 2 (ed. D.Touretzky), Morgan Kaufmann Pub., 1990, 638-641.

Robustness of Feedforward Neural Networks*

Ching-Tai Chiu, Kishan Mehrotra, Chilukuri K. Mohan, and Sanjay Ranka†
4-116 CST, School of Computer and Information Science
Syracuse University, Syracuse, New York 13244-4100
315-443-2368, *email: cchiu/kishan/mohan/ranka@top.cis.syr.edu*

***Abstract*— Many artificial neural networks in practical use can be demonstrated not to be fault tolerant; this can result in disasters when localized errors occur in critical parts of these networks. In this paper, we develop methods for measuring the sensitivity of links and nodes of a feedforward neural network and implement a technique to ensure the development of neural networks that satisfy well-defined robustness criteria. Experimental observations indicate that performance degradation in our robust feedforward network is significantly less than a randomly trained feedforward network of the same size by an order of magnitude.**

I. Introduction

Artificial neural network applications with no built-in or proven fault tolerance can be disastrously handicapped by localized errors in critical parts. Many researchers have assumed that neural networks that contain a large number of nodes and links are fault tolerant. This assumption is unfounded because networks are often trained using algorithms whose only goal is to minimize error. Classical neural learning algorithms such as backpropagation make no attempt to develop fault tolerant neural networks. The existence of redundant resources is only a precondition and does not ensure robustness. Little research has focused explicitly on increasing the fault tolerance of commonly used neural network models of non-trivial size, although the importance of this problem has been recognized [2, 3, 6, 9].

In this paper, we propose techniques to ensure the development of feedforward neural networks [7] that satisfy well-defined robustness criteria. Faults occurring in the training phase may increase training time but are unlikely to affect the performance of the system, because training will continue until faulty components are compensated for by non-faulty parts. If faults are detected in the testing phase, retraining with the addition of new resources can solve the problem. Such a repair would not be possible after system development is complete or if faults occur in a neural network application that has already been installed and is in use. This necessitates robust design of neural networks, for graceful degradation in performance without the need to retrain networks.

Different evaluation measures may be needed for neural networks intended to perform different tasks. Karnin [4] suggests the use of a sensitivity index to determine the extent to which the performance of a neural network depends on a given node or link in the system; for a given failure, Carter et al. [3] measure network performance in terms of the error in function approximation; and Stevenson et al. [8] estimate the probability that an output neuron makes a decision error, for "Madalines" (with discrete inputs).

Our methodology can be briefly summarized as follows. Given a well-trained network, we first eliminate all "useless" nodes in hidden layer(s). We retrain this reduced network, and then add some redundant nodes to the reduced network in a systematic manner, achieving robustness against changes in weights of links that may occur over a period of time.

In Section II, we describe our terminology and measures of robustness. Methods to achieve a robust network are described in Section III. Section IV contains experimental results, and conclusions are discussed in the final section.

*This research was supported by USAF contract F30602-92-C-0031.
†Supported in part by NSF Grant CCR-9110812.

II. Definitions

We consider feedforward $I-H-O$ neural networks, with I input nodes, H nodes in one hidden layer, and O output nodes. The vector of all weights (of the trained network) is denoted by $W=(w_1,\ldots,w_K)$. If the i^{th} component of W is modified by a factor α (i.e., w_i is changed to $(1+\alpha)w_i$) and all other components remain fixed, then the new vector of weights is denoted by $W(i,\alpha)=(w_1,\ldots,(1+\alpha)w_i,\ldots,w_K)$. For a given weight vector W, $E(W)$ denotes the mean square error of the network over the training set, and $E_R(W)$ denote the mean square error over the test set R. The effect on MSE of changing W to $W(i,\alpha)$ is measured in terms of the difference

$$s(i,\alpha)=E(W(i,\alpha))-E(W) \tag{1}$$

or in terms of the partial derivative of MSE with respect to the magnitude of weight change

$$\hat{s}(i,\alpha)=\frac{E(W(i,\alpha))-E(W)}{|w_i\times\alpha|}. \tag{2}$$

If $E(W(i,\alpha))<E(W)$, then a better set of weights must have been accidentally obtained by perturbing W, and retraining can occur for $W(i,\alpha)$. The relative change, α, is allowed to take values from a nonempty finite set A containing values in the range -1 to 1.

Definition 1 Link sensitivity: *Two possible definitions for the sensitivity of the i^{th} link ℓ_i are:*

$$S_\ell(i)=\frac{1}{|A|}\sum_{\alpha\in A}s(i,\alpha) \tag{3}$$

$$\hat{S}_\ell(i)=\frac{1}{|A|}\sum_{\alpha\in A}\hat{s}(i,\alpha). \tag{4}$$

To compute the sensitivity of each link in a network, all weights of the trained network are frozen except the link that is being perturbed with a fault. $E(W)$ is already known and $E(W(i,\alpha))$ can easily be obtained in one feedforward computation with faulty links.

Definition 2 Node sensitivity: *Let $\mathcal{I}_I(j)$ denote the set of all incoming links incident on the j^{th} hidden node, n_j, from the input nodes; let $\mathcal{I}_O(j)$ denote the set of outgoing links from n_j, and let $\mathcal{I}(j)=\mathcal{I}_I(j)\cup\mathcal{I}_O(j)$. Two definitions are possible for node sensitivity:*

(i) *A-sensitivity (average sensitivity) of a node, n_j, is*

$$S_n(j)=\frac{1}{|\mathcal{I}(j)|}\sum_{i\in\mathcal{I}(j)}S_\ell(i). \tag{5}$$

$$\hat{S}_n(j)=\frac{1}{|\mathcal{I}(j)|}\sum_{i\in\mathcal{I}(j)}\hat{S}_\ell(i). \tag{6}$$

(ii) *M-sensitivity (maximal sensitivity) of a node, n_j, is*

$$S_n^*(j)=\max_{i\in\mathcal{I}(j)}S_\ell(i). \tag{7}$$

$$\hat{S}_n^*(j)=\max_{i\in\mathcal{I}(j)}\hat{S}_\ell(i). \tag{8}$$

Definition 3 *The sensitivity S_N of a network N is $\max_{j\in HL(N)}\{S_n(j)\}$, where $HL(N)$ is the set of hidden layer nodes in N.*

III. Addition/Deletion Procedure

In this section, we present a procedure (Figure 1) to build robust neural networks that withstand individual link weight changes: we eliminate unimportant nodes, retrain the reduced network, then add redundant nodes. These three steps are repeated until the desired robustness is achieved.

A. Elimination of Unimportant Nodes

In practice, many of the nodes in a large network serve no useful purpose, and traditional network training algorithms do not ensure that redundant nodes improve fault tolerance. Once a network has been trained, the importance of each hidden layer node can be measured in terms of its sensitivity. Given a reference sensitivity ϵ, node n_j is removed from the hidden layer if $S_n(j)\leq\epsilon$. The value of ϵ can be adjusted such that elimination of all such nodes makes little difference in the performance of the reduced network compared to the original network. In our experiments, we have used $\epsilon=10\%$ of the maximum node sensitivity. The deletion of 'unimportant' nodes (with a small sensitivity) results in an I-H^*-O network that should perform almost as well as the original network. We have observed that H^* is sometimes considerably smaller than H.

Let $\mathcal{TR}$ be the training set and $\mathcal{TS}$ be the test set. Obtain a well-trained weight vector $\mathcal{W}_0$ by training an I-H-O network $\mathcal{N}_0$ on $\mathcal{TR}$.

$i = 0$, and $H^* = H$.
while *terminating-criterion* is unsatisfied **do**
  $\varepsilon = S_N(\mathcal{N}_i) \times 0.1$
  $\mathcal{N}_{i+1} = \mathcal{N}_i - \{n_j | S_n(n_j) < \varepsilon\}$
  $\mathcal{W}_{i+1} = \mathcal{W}_i - \{$all links connected to $n_j\}$
  Retrain the network $\mathcal{N}_{i+1}$.
  $H^* = H^* + 1$
  $\mathcal{N}_{i+1} = \mathcal{N}_i \cup \{n_{H^*}\}$
  $\mathcal{W}_{i+1}$ = Setting the weights of links incident on the new node n_{H^*}, and modifying those connected to the most sensitive node in $\mathcal{N}_i$.
  $i = i + 1$
end while

Figure 1: Addition/Deletion procedure for improved fault tolerance. The *terminating-criterion* is described in section III-C. $S_N(\mathcal{N}_i)$ is the worst case node sensitivity of the network $\mathcal{N}_i$. The procedure for updating the second $\mathcal{W}_{i+1}$ is described in section III-C.

B. Retraining of Reduced Network

Removal of unimportant nodes from the network is expected to make little difference in the resulting MSE. But the MSE of the resulting network with fewer weights may not be in a (local) minimum. In general, if $(x_1, ..., x_n)$ is a local minimum of a function $f^{(n)}$ of n arguments, there is no guarantee that $(x_1, ..., x_{n-i})$ is a local minimum of a function $f^{(n-i)}$ defined as $f^{(n-i)}(x_1, ..., x_{n-i}) \equiv f^{(n)}(x_1, ..., x_{n-i}, 0, ..., 0)$. For our problem, $f^{(n)}$ and $f^{(n-i)}$ are the MSE functions over networks of differing sizes, where the smaller one is obtained by eliminating some parameters of the larger network.

Retraining the reduced network will change the MSE to a (local) minimum. In our experiments, we have observed that the number of iterations needed to retrain the network to achieve the previous level of MSE is usually small (< 10 in most cases).

C. Addition of Redundant Nodes

To enhance robustness, our method is to add extra hidden nodes, in such a way that they share the tasks of the critical nodes—nodes with "high" sensitivity.

Let $w_{i,k}$ denote the weight from the k^{th} input node to the i^{th} hidden layer node, and let $v_{i,k}$ denote the weight from the k^{th} hidden node to the i^{th} output node. Let the j^{th} hidden node have the highest sensitivity, in a I-H^*-O network. Let $h = H^*$. Then the new network is obtained by adding an extra $(h+1)^{th}$ hidden node. The duties of the sensitive node are now shared with this new node. This is achieved by setting up the weights on the new node's links as defined by:

(1) First layer of weights: $w_{h+1,i} = w_{j,i}, \forall i \in I$, {the new node has the same output as the j^{th} node}

(2) Second layer of weights: $v_{k,h+1} = \frac{1}{2} v_{k,j}, \forall k \in O$, {sharing the load of the j^{th} node}

(3) Halving the sensitive node's outgoing link weights $v_{k,j}, \forall k \in O$.

In other words, the first condition guarantees that the outputs of hidden layer nodes n_j and n_{H^*+1} are identical, whereas the second condition ensures that the importance of these two nodes is equal, without changing the network outputs.

After adding the node n_{H^*+1}, node sensitivities are re-evaluated and another node, n_{H^*+2}, is added if the sensitivity of a node is found to be 'too' large. On the other hand, a node is removed if its sensitivity is 'too' low. Our primary criteria for sensitivity of a link and a node are equations (3) and (7). In our experiments, we have found that there is not much difference in the results obtained using the other definitions of sensitivity. A node is deleted if its sensitivity is less than 10% of the sensitivity of the most critical node.

We continue to add nodes until the termination criterion is satisfied, i.e., the improvement in the network's robustness is negligible. We have experimented with two termination criteria. The first criterion is adding extra nodes until the sensitivity of the current most critical node is less than some proportion of the sensitivity of the initial most critical node. The second criterion is adding extra nodes until the number of nodes is equal to the original number of nodes, in order to compare two networks of the same size.

Notation: Let $E(N^{\alpha}_{mn})$ denote the error obtained when the link weight ν_{mn} in the network N is replaced by $(1+\alpha)\nu_{mn}$. Similarly, let $E(N^{\alpha}_{k})$ denote the average error obtained when each of the link weights ν_{mk} in the network N is replaced by $(1+\alpha)\nu_{mk}$. Note that ν_{ij} denotes the weight on the link from the j^{th} hidden node to the i^{th} output node.

Theorem: Let N be a well-trained[1] I-h-O network in which link ν_{ij} is more sensitive than every other link in the second (ν) layer, where sensitivity is defined as the additional error resulting from perturbation of any ν_{mn} to $(1+\alpha)\nu_{mn}$, for some α (i.e., with the singleton perturbation set $A = \{\alpha\}$) such that these perturbations degrade performance, (i.e., the error $E(N) < \max_{m,n}\{E(N^{\alpha}_{mn})\}$. Let M be the network obtained by adding a redundant $(h+1)^{st}$ hidden node to M and adjusting weights of links attached to this new node and to the j^{th} hidden node, as specified in the addition/deletion algorithm given earlier. Then M is more robust than N, i.e., the sensitivity of M is lower than that of N.

The above theorem pertained to the special case where A was a singleton set. This result extends to the case when A contains many elements, and for node faults; these additional results and proofs are omitted due to lack of space. The theorem holds even with minor variations in the definitions of the sensitivity. A premise of the above theorem is that perturbations should degrade performance: if such is not the case, i.e., if network error actually decreases as a result of introducing "faults" into the system, then we replace the network by the new 'perturbed' network with better performance, which is then retrained.

IV. Experimental Evaluation

We evaluate our algorithm by comparing the sensitivity of the original network (with redundant nodes, randomly trained using the traditional backpropagation algorithm) with that of the network evolved using our proposed algorithm. Robustness of a network is measured in terms of graceful degradation in MSE and MIS(fraction of misclassification errors) on the test set and the training set. In the average cases, we plot the sets AC_{mse} and AC_{mis}, whereas in the worst cases we plot the sets WC_{mse} and WC_{mis}, which are defined as follows, where $\mathcal{I}$ is the set of all links and $C_R(W)$ is the fraction of misclassification errors on test set.

- $AC_{mse} = \{(x,y) | -100 \le x \le 100, x \bmod 5 \equiv 0, y = \frac{1}{|\mathcal{I}|}\sum_{i\in\mathcal{I}} E_R(W(i, \frac{x}{100}))\}$,
- $WC_{mse} = \{(x,y) | -100 \le x \le 100, x \bmod 5 \equiv 0, y = \max_{i\in\mathcal{I}} E_R(W(i, \frac{x}{100}))\}$,
- $AC_{mis} = \{(x,y) | -100 \le x \le 100, x \bmod 5 \equiv 0, y = \frac{1}{|\mathcal{I}|}\sum_{i\in\mathcal{I}} C_R(W(i, \frac{x}{100}))\}$,
- $WC_{mis} = \{(x,y) | -100 \le x \le 100, x \bmod 5 \equiv 0, y = \max_{i\in\mathcal{I}} C_R(W(i, \frac{x}{100}))\}$.

We performed four series of experiments for each problem using the following combinations of sensitivity definitions, where A is the set of values by which a weight is perturbed, when testing sensitivity.

Combination 0: Eq. (7) and $A = \{-1\}$.

Combination 1: Eq. (8) and $A = \{-1\}$.

Combination 2: Eq. (8) and $A = \{+0.1, -0.1\}$.

Combination 3: Eq. (8) and $A = \{\pm 1, \pm\frac{1}{2}\}$.

Combinations 0, 1, and 3 have almost identical performance, and also perform better than combination 2, possibly because the former measure performance for large changes in weights. Experimental results are shown only for combination 0, and only for one classic 3-class classification problem: Fisher's Iris data, with a four-dimensional input vector for each pattern, input values being rescaled to fall between 0 and 1. A third of the data points were not used for training, and were used as a test set. To start with, we trained a 4-10-3 neural network. Algorithm 1 reduced it to a 4-4-3 network in the first deletion step and then it was built up, successively, to a 4-10-3 network. There were 10 hidden nodes in the original network; our criterion reduced it to 4, then increased it to 10 as described in Table 1. When a 9-node network was obtained in this manner and retrained, two nodes could again be removed due to the sensitivity criteria, and more nodes were then added following our algorithm in Figure 1. The original 10-node network was roughly as robust as a 6-node network for high perturbations, and worse than all other cases for small perturbations.

Performance degradations of the initial and final 4-10-3 networks are shown in Table 1, and Figures 2 and 3 for the test set. Our robustness procedure achieved 83% improvement on average sensitivity and 81% improvement on worst sensitivity for this problem.

[1] "Well-trained" means that the network error is almost 0.

	Initial Net	Final Net
Hidden nodes	10	10
Training MSE	0.005130	0.005129
Testing MSE	0.025139	0.022123
Training Correctness(%)	99.00	99.00
Testing Correctness(%)	94.00	96.00
Avg. Sensitivity	0.025686	0.004314
Wst. Sensitivity	0.110977	0.021464

Table 1: Results of Fisher's Iris data on Combination 0. Deletion/addition process is $10 \rightarrow 4 \rightarrow 5 \rightarrow 6 \rightarrow 7 \rightarrow 8 \rightarrow 9 \rightarrow 7 \rightarrow 8 \rightarrow 9 \rightarrow 10$.

V. Discussion

We have developed a procedure for improving the robustness of feedforward neural networks, and shown that our algorithm results in significantly large improvements in the fault tolerance of networks trained for two multiclass classification problems. Minor variants of the algorithm, e.g., in using additional retraining steps in the body of the while-loop, resulted in no significant improvement.

Another possible approach to improve robustness is to build into the training algorithm a mechanism to discourage large weights: instead of the mean square error E_0, the new quantity to be minimized is of the form E_0+ (a penalty term monotonically increasing with the size of weights). We implemented three different possibilities, modifying each weight w_i by the quantities $-\eta(\frac{\partial E_0}{\partial w_i} + cw_i)$, $-\eta\frac{\partial E_0}{\partial w_i}(1+cw_i)$, and $-\eta\left(\frac{\partial E_0}{\partial w_i}(1+\frac{1}{2}c\sum_i w_i^2) + cw_iE_0\right)$, respectively, for many different values of c. This approach does improve robustness slightly, when compared to plain backpropagation, but the results are much less impressive than with our addition/deletion procedure. We also performed experiments combining both approaches, and this resulted in very slight improvements over the addition/deletion procedure.

References

[1] T.Anderson and P.A.Lee, Fault Tolerance: Principles and Practice, Prentice-Hall, 1981.

[2] L.A.Belfore, Fault Tolerance of Neural Networks, Ph.D. Dissertation, Univ. of Virginia, 1989.

[3] M.J. Carter, F.J.Rudolph and A.J. Nucci, *Operational Fault Tolerance of CMAC Networks*, Proc. Workshop on Neural Info. Proc. Systems, 1989, 340-347.

[4] E.D. Karnin, *A Simple Procedure for Pruning Back-Propagation Trained Neural Networks*, IEEE Trans. Neural Networks 1 (2), June 1990, 239-242.

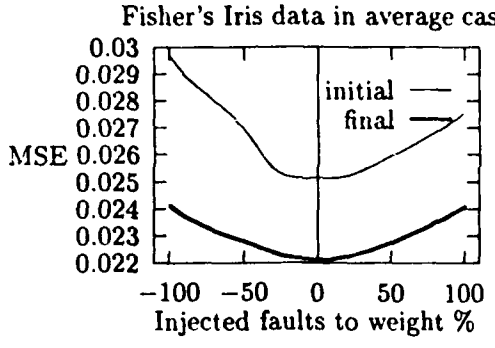
[5] A. Krogh, J.A. Hertz, *A Simple Weight Decay Can Improve Generation.*

[6] T. Petsche and B.W. Dickinson, *Trellis Codes, Receptive Fields, and Fault Tolerant, Self-Repairing Neural Networks*, IEEE Trans. Neural Networks 1 (2), June 1990, 154-166.

[7] D.E.Rumelhart, J.L.McClelland, and the PDP Research Group, *Parallel Distributed Processing*, Vol.1, 1986.

[8] M.Stevenson, R.Winter and B.Widrow, *Sensitivity of Layered Neural Networks to Errors in the Weights*, Proc. Int'l. Joint Conf. Neural Networks, San Diego, June 1990, vol.I:337-340.

[9] G. Swaminathan, S. Srinivasan and S. Mitra, *Fault Tolerance in Neural Networks*, Proc. Int'l. Joint Conf. Neural Networks, San Diego, June 1990, vol.II:699-702.



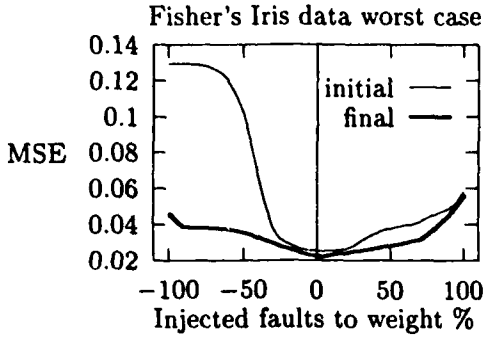
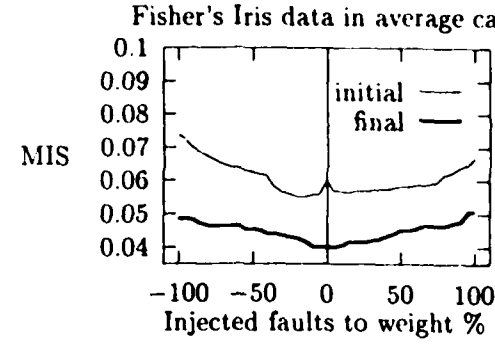


Figure 2: Degradation in mean square error for the test set, using networks with 10 hidden nodes, trained on Fisher's Iris data.



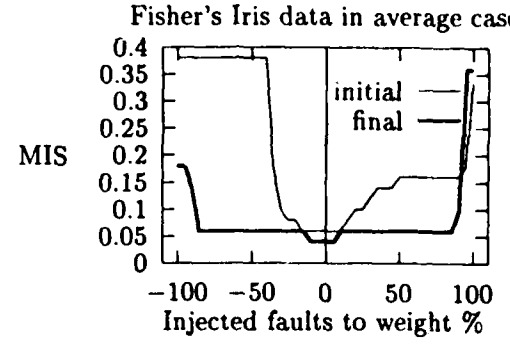


Figure 3: Degradation in the number of misclassified samples for the test set, using networks with 10 hidden nodes, trained on Fisher's Iris data.

*U.S. GOVERNMENT PRINTING OFFICE: 1994-510-117-50003

MISSION

OF

ROME LABORATORY

Mission. The mission of Rome Laboratory is to advance the science and technologies of command, control, communications and intelligence and to transition them into systems to meet customer needs. To achieve this, Rome Lab:

a. Conducts vigorous research, development and test programs in all applicable technologies;

b. Transitions technology to current and future systems to improve operational capability, readiness, and supportability;

c. Provides a full range of technical support to Air Force Materiel Command product centers and other Air Force organizations;

d. Promotes transfer of technology to the private sector;

e. Maintains leading edge technological expertise in the areas of surveillance, communications, command and control, intelligence, reliability science, electro-magnetic technology, photonics, signal processing, and computational science.

The thrust areas of technical competence include: Surveillance, Communications, Command and Control, Intelligence, Signal Processing, Computer Science and Technology, Electromagnetic Technology, Photonics and Reliability Sciences.